Günter Schmitt

Maschinenorientierte Programmierung für Mikroprozessoren

2., verbesserte Auflage

Mit 321 Bildern, 125 Tabellen
und 246 Programmen

Reihe Datenverarbeitung
R. Oldenbourg Verlag München Wien

Reihe Datenverarbeitung

Herausgeber:

Klaus Becker-Berke, Rudolf Herschel,
Wolfgang Hilberg, Robert Piloty

363

Maschinenorientierte Programmierung für Mikroprozessoren

von Dipl.-Ing. **Günter Schmitt,**
Fachhochschule der Deutschen Bundespost,
Dieburg

Mit 321 Bildern, 125 Tabellen
und 246 Programmen

2., verbesserte Auflage

R. Oldenbourg Verlag München Wien 1980

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Schmitt, Günter:
Maschinenorientierte Programmierung für Mikroprozessoren / von Günter Schmitt. – 2., verb. Aufl. – München, Wien : Oldenbourg, 1980.
(Reihe Datenverarbeitung)
ISBN 3-486-22892-7

© 1980 R. Oldenbourg Verlag GmbH, München

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege sowie der Speicherung und Auswertung in Datenverarbeitungsanlagen, bleiben auch bei auszugsweiser Verwertung vorbehalten. Werden mit schriftlicher Einwilligung des Verlages einzelne Vervielfältigungsstücke für gewerbliche Zwecke hergestellt, ist an den Verlag die nach § 54 Abs. 2 Urh.G. zu zahlende Vergütung zu entrichten, über deren Höhe der Verlag Auskunft gibt.

Druck: Weber Offset, München
Bindearbeiten: R. Oldenbourg Graphische Betriebe GmbH, München

ISBN 3-486-22892-7

Inhaltsverzeichnis

1. Wie arbeitet ein Mikrocomputer? 9

1.1 Darstellung der Daten im Mikrocomputer. 12
1.2 Zahlensysteme und Umrechnungsverfahren 14
1.3 Rechenschaltungen 19
1.4 Die Arithmetisch-Logische Einheit (ALU) 27
1.5 Speicherschaltungen. 29
1.6 Datenspeicher. 37
1.7 Ein einfaches Rechenwerk mit Datenspeicher und Ein-/Ausgaberegistern 39
1.8 Ein einfaches Steuerwerk mit Programmspeicher 42
1.9 Ein einfacher Modell-Mikrocomputer 45

2. Der Aufbau ausgeführter Mikrocomputer 51

2.1 Der Mikrocomputer KIM-1 52
2.2 Ein Mikrocomputer mit dem Prozessor Motorola 6800. 56
2.3 Mikrocomputer mit den Prozessoren INTEL 8080 und 8085 59
2.4 Die Programmierung von Mikrocomputern 63
2.5 Das Programmieren in Assemblerschreibweise 66
2.5.1 Eine vereinfachte Assemblersprache für den KIM-1 68
2.5.2 Eine vereinfachte Assemblersprache für den 6800 70
2.5.3 Eine vereinfachte Assemblersprache für den 8080/8085 72

3. Der Datentransport 75

3.1 Der Datentransport zwischen Registern des Prozessors 75
3.1.1 Die Register des KIM-1. 75
3.1.2 Die Register des Motorola 6800 77
3.1.3 Die Register des 8080/8085. 78
3.2 Das Laden von Registern. 80
3.2.1 Das Laden der KIM-1-Register 81
3.2.2 Das Laden der 6800-Register 84
3.2.3 Das Laden der 8080/8085-Register 87

3.3 Das Speichern von Registern 89
3.3.1 Das Speichern der KIM-1-Register. 89
3.3.2 Das Speichern der 6800-Register 90
3.3.3 Das Speichern der 8080/8085-Register 92
3.4 Der Datentransport zwischen Speicherstellen 93
3.5 Die symbolische Darstellung der Transportbefehle und Transportvorgänge. 93
3.6 Einführung in die Programmierung von Parallelschnittstellen 95
3.6.1 Programmierung des Bausteins 6530 (KIM-1) 96
3.6.2 Die Programmierung der Parallelschnittstelle 6820. 98
3.6.3 Die Programmierung der Parallelschnittstelle 8255. 100
3.6.4 Die Programmierung der Parallelschnittstelle 8155. 103
3.6.5 Die Programmierung der Parallelschnittstelle 8355. 104

4. Die Sprungbefehle und ihre Anwendungen in Schleifen . . . 106
4.1 Der Ablauf einer Befehlsfolge 106
4.2 Der unbedingte Sprung. 108
4.2.1 Der unbedingte Sprungbefehl des KIM-1 110
4.2.2 Die unbedingten Sprungbefehle des 6800. 111
4.2.3 Der unbedingte Sprungbefehl des 8080/8085 114
4.3 Der bedingte Sprung 115
4.3.1 Die bedingten Sprünge des KIM-1 117
4.3.2 Die bedingten Sprünge des 6800. 121
4.3.3 Die bedingten Sprünge des 8080/8085. 123
4.4 Zählbefehle und Zählschleifen 125
4.4.1 Die Zähl- und Vergleichsbefehle des KIM-1. 129
4.4.2 Die Zähl- und Vergleichsbefehle des 6800 132
4.4.3 Die Zähl- und Vergleichsbefehle des 8080/8085 136
4.5 Die Berechnung von Datenadressen über Indexregister 142
4.5.1 Adreßrechnungen mit den Indexregistern des KIM-1 143
4.5.2 Adreßrechnungen mit dem Indexregister des 6800. 149
4.5.3 Die Registerpaar-Adressierung des 8080/8085. 157
4.6 Die indirekte Adressierung 163
4.6.1 Die indirekte Adressierung des KIM-1 164
4.6.2 Indirekte Adressierung mit dem 6800 167
4.6.3 Indirekte Adressierung mit dem 8080/8085 169

5. Programmverzweigungen 171
5.1 Verzweigungen mit Sprung- und Vergleichsbefehlen. 172
5.1.1 Verzweigungen mit Vergleichsbefehlen des KIM-1 173
5.1.2 Verzweigungen mit Vergleichsbefehlen des 6800. 177
5.1.3 Verzweigungen mit Vergleichsbefehlen des 8080/8085. 181

5.2 Verzweigungen mit Hilfe von logischen Befehlen 185
5.2.1 Verzweigungen mit logischen Befehlen des KIM-1 186
5.2.2 Verzweigungen mit logischen Befehlen des 6800 187
5.2.3 Verzweigungen mit logischen Befehlen des 8080/8085 188
5.3 Verzweigungen mit Hilfe von Schiebebefehlen 188
5.3.1 Verzweigungen mit Schiebebefehlen des KIM-1 191
5.3.2 Verzweigungen mit Schiebebefehlen des 6800 193
5.3.3 Verzweigungen mit Schiebebefehlen des 8080/8085 195
5.4 Der berechnete (indirekte) Sprung 197
5.4.1 Der indirekte Sprung des KIM-1 200
5.4.2 Der indirekte Sprung mit dem 6800 201
5.4.3 Der indirekte Sprung mit dem 8080/8085 203

6. Die Verarbeitung von Daten 206
6.1 Die logischen Befehle 206
6.1.1 Die logischen Befehle des KIM-1 209
6.1.2 Die logischen Befehle des 6800 211
6.1.3 Die logischen Befehle des 8080/8085 213
6.2 Die Schiebebefehle 215
6.2.1 Die Schiebebefehle des KIM-1 217
6.2.2 Die Schiebebefehle des 6800 219
6.2.3 Die Schiebebefehle des 8080/8085 221
6.3 Die arithmetischen Befehle 222
6.3.1 Die arithmetischen Befehle des KIM-1 223
6.3.2 Die arithmetischen Befehle des 6800 229
6.3.3 Die arithmetischen Befehle des 8080/8085 238
6.4 Erweiterung des Zahlenbereiches 248
6.4.1 Beispiele für den KIM-1 249
6.4.2 Beispiele für den 6800 251
6.4.3 Beispiele für den 8080/8085 253
6.5 Multiplikation und Division 255
6.5.1 Beispiele für den KIM-1 262
6.5.2 Beispiele für den 6800 265
6.5.3 Beispiele für den 8080/8085 266
6.6 Die Arbeit mit Tabellen 268

7. Unterprogramme und Programmunterbrechungen 276
7.1 Die Stapelbefehle 276
7.1.1 Die Stapelbefehle des KIM-1 277
7.1.2 Die Stapelbefehle des 6800 280
7.1.3 Die Stapelbefehle des 8080/8085 283

7.2 Unterprogrammtechnik ... 287
7.2.1 Die Unterprogrammbefehle des KIM-1 ... 292
7.2.2 Die Unterprogrammbefehle des 6800 ... 294
7.2.3 Die Unterprogrammbefehle des 8080/8085 ... 296
7.3 Programmunterbrechungen durch äußere Signale (Interrupt) ... 298
7.3.1 Die Interrupt-Möglichkeiten des KIM-1 ... 299
7.3.2 Die Interrupt-Möglichkeiten des 6800 ... 303
7.3.3 Die Interrupt-Möglichkeiten des 8080 ... 309
7.3.4 Die Interrupt-Möglichkeiten des 8085 ... 312

8. Lösungen der Übungsaufgaben ... 316

9. Ergänzende und weiterführende Literatur ... 334

10. Anhang ... 336

11. Register ... 346
11.1 Allgemeiner Teil ... 346
11.2 KIM-1-Teil ... 347
11.3 6800-Teil ... 348
11.4 8080/8085-Teil ... 349

1. Wie arbeitet ein Mikrocomputer?

Der Mikrocomputer, auch Mikrorechner genannt, gehört mit dem Taschenrechner, dem Großrechner (Computer) und der digitalen Steuerungs-Schaltung zu den datenverarbeitenden Geräten, die

- aus elektronischen Schaltungen mit Widerständen, Dioden, Transistoren usw. aufgebaut sind,
- digital arbeiten, d.h. mit den beiden elektrischen Zuständen "niedriges Potential" und "hohes Potential",
- Daten (Zahlen, Zeichen und Signale) aufnehmen, speichern, verarbeiten sowie ausgeben und
- sehr schnell und zuverlässig arbeiten.

Zunächst soll der Mikrocomputer von den drei anderen Geräten abgegrenzt werden.

Der Mikrocomputer ist kein Taschenrechner (Bild 1), der zu einem unentbehrlichen Rechenknecht des Ingenieurs geworden ist. Beide arbeiten mit ähnlichen Rechen- und Speicherschaltungen und werden in der gleichen Halbleitertechnik hergestellt. Während jedoch der Taschenrechner z.B. nach dem Drücken der Divisionstaste zwei Zahlen durcheinander dividiert, müßte der Anwender eines Mikrocomputers erst ein Programm zur Division schreiben und im Mikrocomputer speichern. Es wäre jedoch durchaus möglich, aber nicht sinnvoll, einen Mikrocomputer mit Eingabetastatur, Zahlenanzeige und Rechenprogrammen zu entwickeln, der wie ein Taschenrechner arbeitet.

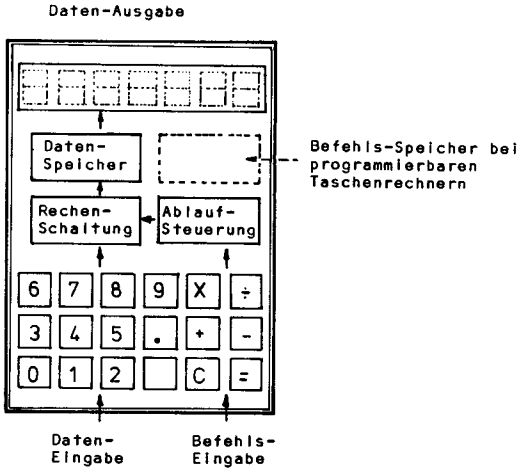


Bild 1: Aufbau eines Taschenrechners

Der Mikrocomputer ist auch kein Großrechner (Bild 2), der je nach eingespeichertem Programm Bußgeldbescheide oder Telefonrechnungen druckt, das Profil eines Flugzeugflügels berechnet oder den Start einer Mondrakete überwacht. Beiden gemeinsam ist ein Speicher zur Aufnahme der Befehle und Daten, das Steuerwerk für die Ausführung der Befehle und das Rechenwerk, das die Daten verarbeitet. Im Gegensatz zum allgemein programmierbaren und einsetzbaren Großrechner ist der Mikrocomputer meist Bestandteil eines Gerätes, wenn er z.B. die Arbeit einer Werkzeugmaschine steuern soll; das Programm ist fest gespeichert und wird nur in Ausnahmefällen geändert. Der Mikrocomputer wird in zunehmendem Maße auch in "intelligente" Büromaschinen eingebaut, die für bestimmte Aufgaben frei programmierbar sind. Bei dieser Anwendung gerät er in den Einsatzbereich des Computers.

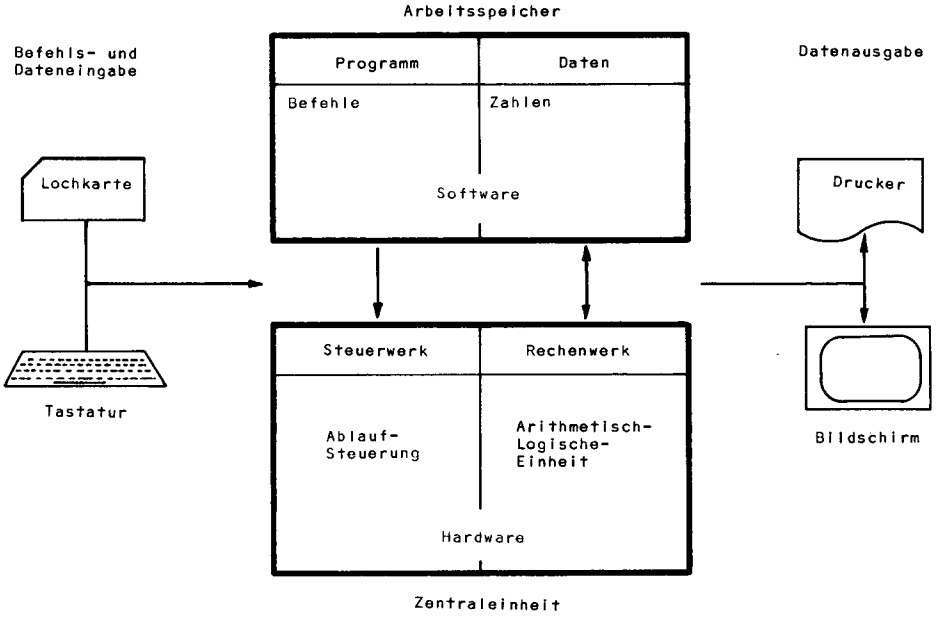


Bild 2: Aufbau eines Computers (Großrechner)

Das Hauptanwendungsgebiet des Mikrocomputers ist die Steuerung von Geräten und Anlagen, wie z.B. Waagen, Registrierkassen, Verkehrsampeln, Werkzeugmaschinen und Aufzügen. Dabei ersetzt er in zunehmendem Maße digitale Steuerungs-Schaltungen, die bisher aus einzelnen Bauelementen wie z.B. Bausteinen der TTL-Serie SN74XXX aufgebaut waren. Der Entwicklungsingenieur setzte eine solche Schaltung aus vielen verschiedenen Bauelementen (Logikgatter, Register, Rechenschaltungen) zusammen, für jedes Gerät entstand eine neue Kombination von Bauelementen und deren Verdrahtung.

Der Mikrocomputer nach Bild 3 ist eine Baugruppe in einem von ihm zu steuernden Gerät. Er besteht aus mindestens vier Bauelementen:

1. Ein Nur-Lese-Speicher (ROM) enthält das Programm mit den Befehlen und konstanten Daten.

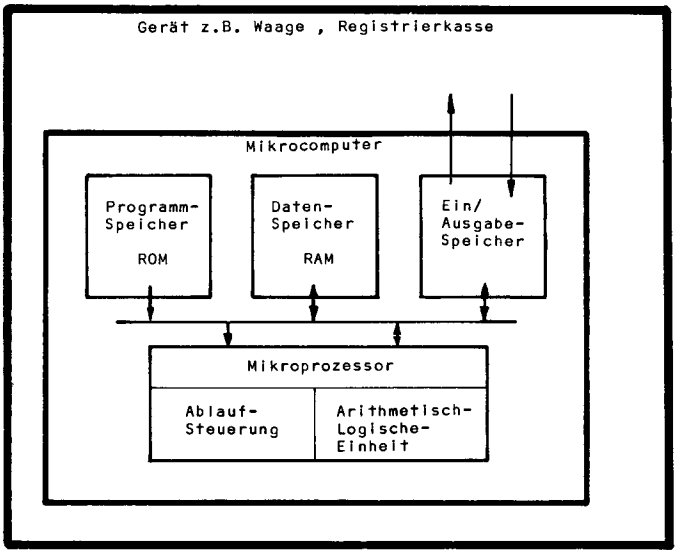


Bild 3: Aufbau eines Mikrocomputers

2. Ein Schreib-Lese-Speicher (RAM) nimmt die variablen Daten auf, die z.B. als Eingabewerte oder Zwischenergebnisse auftreten.
3. Der Ein/Ausgabe-Speicher stellt mit seinen Anschlußleitungen die Verbindung zum Gerät her.
4. Der Mikroprozessor besteht aus einer Ablaufsteuerung, die die Befehle ausführt, und einem Rechenwerk (Arithmetisch-logische Einheit) zur Verarbeitung der Daten.

Jedes der vier Bauelemente besteht aus einem Siliziumplättchen von ca. 6 mm Kantenlänge; es gibt auch Mikrocomputer, bei denen alle vier Schaltungen in einem einzigen Bauelement untergebracht

Fest verdrahtete Logik	Programmierbare Logik
Viele verschiedene Bauelemente für besondere Anwendungen.	Wenige aber vielseitig anwendbare Bauelemente.
Maßgeschneiderte Lösung ohne überflüssige Funktionen.	Es wird nur ein Teil der Möglichkeiten ausgenutzt.
Änderung der Funktion durch Umbau der Schaltung.	Änderung der Funktion durch Neuprogrammierung, Schaltung bleibt.
Entwickler muss die Digitaltechnik beherrschen.	Entwickler muss die Digitaltechnik beherrschen und programmieren können.
Vorteile bei den Kosten und in der Entwicklungszeit bei kleinen Projekten sowie in der Geschwindigkeit.	Vorteile bei den Kosten und in der Entwicklungszeit bei großen Projekten, die Geräte können "intelligentere" Funktionen übernehmen.

Bild 4: Vergleich verdrahtete - programmierbare Logik

sind. Der Mikrocomputer enthält wie eine digitale Steuerung logische und arithmetische Schaltungen sowie Register, Zähler und Speicherschaltungen. Aber erst das Programm entscheidet, welche dieser Schaltungen zu welchem Zeitpunkt tätig werden soll. Der Anwender hat also nur eine beschränkte Anzahl von Schaltungen zur Verfügung, die er jedoch in beliebiger Reihenfolge beliebig oft benutzen kann. Bild 4 vergleicht den Mikrocomputer mit einer fest verdrahteten Logik.

1.1 Darstellung der Daten im Mikrocomputer

Daten sind Zahlen (z.B. Ziffernanzeige eines Meßwertes), Zeichen (z.B. Buchstaben einer Tastatur) oder analoge Signale (z.B. die Spannung eines Widerstands-Thermometers), die in digitale Werte umgewandelt werden müssen. Die datenverarbeitenden elektronischen Schaltungen arbeiten besonders schnell und zuverlässig, wenn man sie in nur zwei Zuständen betreibt:

Schalter ein	-	Schalter aus
Strom fließt	-	Strom fließt nicht
hohes Potential	-	niedriges Potential

Für diese beiden Zustände verwendet die Digitaltechnik die beiden Bezeichnungen (positive Logik):

H	für hohes Potential (high)
L	für niedriges Potential (low)

Bild 5 zeigt ein Beispiel für den Verlauf einer Spannung, die man durch die Zeichenfolge LHLLLLHH beschreiben kann.

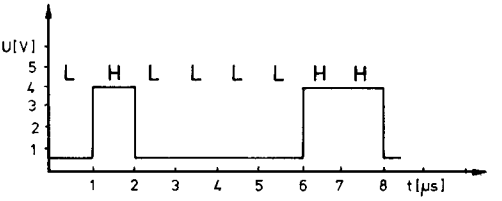


Bild 5: Serieller Datentransport

Die Datenverarbeitung betrachtet nicht die elektrischen Potentiale, sondern ihren logischen Inhalt und verwendet dazu die Zeichen 0 und 1 (positive Logik).

1 = H
0 = L

Der Potentialverlauf des Bildes 5 lautet dann in der Schreibweise der Datenverarbeitung 01000011 . Diese Darstellung heißt binär oder zweiwertig. Eine Speicherstelle, die eines der beiden

Binärzeichen enthält, bezeichnet man als Bit. Acht Bits, also acht Speicherstellen, bilden ein Byte. Alle Daten sind durch die beiden Zeichen 0 und 1 darzustellen.

Für die Darstellung der Zahlen gibt es zwei Möglichkeiten:

1. Jede Ziffer der Dezimalzahl wird in vier bit binär codiert (verschlüsselt).

   Beispiel: 123 = 000100100011

   Ziffer 1 = 0001

   Ziffer 2 = 0010

   Ziffer 3 = 0011

   Bild 6 zeigt den in diesem Beispiel verwendeten Code (Schlüssel).

Ziffer	Codierung
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Bild 6: 4-Bit-Code (8-4-2-1 BCD-Code)

   Der Anhang 1 enthält weitere Codes zur Darstellung von Dezimalziffern. Die Zeichen 0100 0011 des Bildes 5 stellen damit die Dezimalzahl 43 dar.

2. Die Dezimalzahl wird in eine Dualzahl umgerechnet; das duale Zahlensystem kennt nur die Ziffern 0 und 1.

   Beispiel: 123 = 01111011

   Die Zeichen 01000011 des Bildes 5 stellen die Dezimalzahl 67 dar. Die Umrechnungsverfahren folgen im nächsten Abschnitt.

Für die Darstellung von Zeichen verwendet man häufig einen 7-Bit-Code. Fügt man ein achtes Bit als Kontrollbit hinzu, so kann man in einem Byte genau ein Zeichen speichern. Bild 7 zeigt einige Beispiele für diesen Code.

Buchstaben	Code	Ziffern	Code	Sonder-Zeichen	Code
A	01000001	0	00110000		00100000
B	01000010	1	00110001	!	00100001
C	01000011	2	00110010	"	00100011
D	01000100	3	00110011	#	00100100
.	.	.	.	.	.
.	.	.	.	.	.

Bild 7: 7-Bit-Zeichencode (ASCII)

Der Anhang 2 enthält die vollständige Tabelle des ASCII-Codes. Die Zeichen 0100 0011 des Bildes 5 stellen den Buchstaben C dar.

Die analogen Eingangs-Signale (Spannungen und Ströme) werden durch Wandler in digitale Signale umgesetzt. Die digitalen Ausgangs-Signale des Mikrocomputers können wieder in analoge Spannungen und Ströme verwandelt werden. Die Zeichen 01000011 des Bildes 5 könnten z.B. eine Spannung von 67 mV darstellen.

1.2 Zahlensysteme und Umrechnungsverfahren

Das dezimale Zahlensystem verwendet die zehn Ziffern von 0 bis 9. Der Wert einer Ziffer hängt von der Stelle ab, an der sie in der Ziffernfolge steht. Die letzte Stelle hat den Wert 1 = 10^0, die vorletzte den Wert 10 = 10^1 und die davorliegende den Wert 100 = 10^2. Die Wertigkeiten sind Potenzen zur Basis 10.

Beispiel: $123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$
$= 100 + 20 + 3$

Multipliziert man jede Ziffer mit ihrer Wertigkeit und addiert man die Produkte, so ergibt sich wieder die ursprüngliche Zahl.

Das duale Zahlensystem verwendet die beiden Ziffern 0 und 1. Die Basis ist die Zahl 2; damit sind die Wertigkeiten der Dualstellen Potenzen zur Basis 2.

Beispiel:

$$1111011 = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$= 64 + 32 + 16 + 8 + 0 + 2 + 1$$
$$= 123$$

Multipliziert man auch hier jede Ziffer mit ihrer Wertigkeit und addiert die Produkte, so ergibt sich die Zahl im Dezimalsystem.

Umwandlungsregel

> Die Dualstellen sind mit ihrer Wertigkeit zu multiplizieren, die Produkte sind zu addieren und ergeben die Dezimalzahl.

Will man umgekehrt eine Dezimalzahl in eine Dualzahl verwandeln, so ist sie in Zweierpotenzen zu zerlegen.

Beispiel:

Die Dezimalzahl	123	enthält die	64	$= 1 \cdot 2^6$	123 - 64	= 59
Der Rest von	59	enthält die	32	$= 1 \cdot 2^5$	59 - 32	= 27
Der Rest von	27	enthält die	16	$= 1 \cdot 2^4$	27 - 16	= 11
Der Rest von	11	enthält die	8	$= 1 \cdot 2^3$	11 - 8	= 3
Der Rest von	3	enthält nicht	4	$= 0 \cdot 2^2$	3 - 0	= 3
Der Rest von	3	enthält die	2	$= 1 \cdot 2^1$	3 - 2	= 1
Der Rest von	1	enthält die	1	$= 1 \cdot 2^0$	1 - 1	= 0

$$123 = 1\cdot2^6 + 1\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$$
$$= 1111011$$

Das gebräuchlichere Divisionsrestverfahren spaltet die niedrigsten Stellen zuerst ab.

Beispiel:

```
123 : 2 = 61 Rest 1      oder  123 = 2·61 + 1
 61 : 2 = 30 Rest 1      oder   61 = 2·30 + 1
 30 : 2 = 15 Rest 0      oder   30 = 2·15 + 0
 15 : 2 =  7 Rest 1      oder   15 = 2·7  + 1
  7 : 2 =  3 Rest 1      oder    7 = 2·3  + 1
  3 : 2 =  1 Rest 1      oder    3 = 2·1  + 1
  1 : 2 =  0 Rest 1      oder    1 = 2·0  + 1
```

Die Dualzahl ist 1111011.

Umwandlungsregel

> Die Dezimalzahl ist laufend durch 2 zu dividieren. Die Reste (0 oder 1) ergeben die Dualstellen. Bei der ersten Division entsteht die wertniedrigste, bei der letzten die werthöchste Stelle.

Bei Dezimalzahlen läßt man führende Nullen fort, in der Datenverarbeitung arbeitet man oft mit einer festen Zahl von Stellen und besetzt dann auch die höchsten Stellen mit Nullen. Die Dezimalzahl 123 lautet als achtstellige Dualzahl:

01111011

Um Verwechslungen vorzubeugen, setzt man oft die Basis des Zahlensystems als Index hinter die Zahl. Zur Unterscheidung von dem Buchstaben "O" kann die Ziffer "Ø" auch einen Schrägstrich erhalten.

Beispiel: $10_{10} \triangleq 1010_2 \triangleq 1Ø1Ø$

Für Stellen hinter dem Komma gelten ähnliche Regeln. Die Wertigkeiten der Stellen sind negative Potenzen zur Basis des entsprechenden Zahlensystems.

Beispiel:
$$0{,}6875 = 6\cdot10^{-1} + 8\cdot10^{-2} + 7\cdot10^{-3} + 5\cdot10^{-4}$$
$$= 0{,}6 + 0{,}08 + 0{,}007 + 0{,}0005$$

$$0{,}1011_2 = 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} + 1\cdot2^{-4}$$
$$= 1\cdot0{,}5 + 0\cdot0{,}25 + 1\cdot0{,}125 + 1\cdot0{,}0625$$
$$= 0{,}5 + 0{,}125 + 0{,}0625$$
$$= 0{,}6875_{10}$$

Stellen hinter dem Dezimalkomma werden durch Multiplikation mit der Zahl 2 und Abspaltung einer 0 oder 1 in eine Dualzahl umgerechnet.

Beispiel:

```
0,6875 · 2 = 1,3750 = 0,3750 + 1
0,3750 · 2 = 0,7500 = 0,7500 + 0
0,7500 · 2 = 1,5000 = 0,5000 + 1
0,5000 · 2 = 1,0000 = 0,0000 + 1
0,0000 · 2 = 0,0000 = 0,0000 + 0
0,0000 · 2 = 0,0000 = 0,0000 + 0
```

Die Dezimalzahl 0,6875 lautet dual 0,101100

Umwandlungsregel

> Dezimalstellen hinter dem Komma werden fortlaufend mit 2 multipliziert. Ist das Ergebnis größer oder gleich 1, so zieht man die sich ergebende Dualstelle 1 ab. Ist das Ergebnis kleiner 1, so ergibt sich die Dualstelle 0. Der erste Schritt liefert die erste Stelle hinter dem Komma.

Ganz allgemein kann man jede Zahl zur Basis eines Zahlensystems machen. Die Wertigkeiten sind immer Potenzen zur Basis des betreffenden Systems. In den Umrechnungsverfahren ist die Zahl 2 durch die Basis des anderen Zahlensystems zu ersetzen.

Wenn man eine Dezimalzahl in eine Dualzahl verwandelt, so erhöht sich die Zahl der Stellen um etwa das 3,3 fache. Daher faßt man oft mehrere Dualstellen zu einer neuen Ziffer zusammen. Drei Dualstellen ergeben eine Oktalziffer, vier Dualstellen eine Hexadezimalziffer.

Beispiele:

$$123_{10} = 001\ 111\ 011_2 = 1\ \ 7\ \ 3 = 173_8$$

$$173_8 = 1\cdot8^2 + 7\cdot8^1 + 3\cdot8^0 = 64 + 56 + 3$$

$$123_{10} = 0111\ 1011_2 = 7\ \ B = 7B_{16}$$

$$7B_{16} = 7\cdot16^1 + 11\cdot16^0 = 112 + 11$$

Das oktale Zahlensystem verwendet die Ziffern 0 bis 7, die Basis ist 8, die Werte der Stellen sind Potenzen von 8.

Das hexadezimale Zahlensystem verwendet die Ziffern 0 bis 9 und zusätzlich die Zeichen A,B,C,D,E und F.

Damit ergibt sich folgende Tabelle:

binär	hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Es hat damit 16 Ziffern, die Werte der Stellen sind Potenzen von 16. Die acht Bits eines Bytes kann man damit in zwei Hexadezimalziffern ausdrücken. Deshalb bevorzugt man bei Mikrocomputern das Hexadezimalsystem zur Eingabe und Ausgabe von Daten. Die Zeichen 0100 0011 des Bildes 5 lauten hexadezimal 43_{16}.

Negative Zahlen versieht man im Dezimalsystem mit einem negativen Vorzeichen, bei der Addition einer negativen Zahl wird sie subtrahiert; man vertauscht also Vor- und Rechenzeichen.

Beispiel: 44 + (-13) = 44 - (+13) = 31

In der digitalen Rechentechnik will man jedoch nur die geforderte Rechenoperation ohne Rücksicht auf das Vorzeichen der Daten ausführen. Negative Zahlen sind also so darzustellen, daß sie das Rechenwerk ohne Berücksichtigung des Vorzeichens richtig verarbeiten kann. Man kann nun das negative Vorzeichen einer Zahl einfach dadurch beseitigen, indem man eine größere positive Zahl dazuaddiert. Im Beispiel einer zweistelligen Dezimalzahl kann man z.B. die Dezimalzahl 100 nehmen, die in der Lage ist, alle negativen zweistelligen Dezimalzahlen positiv zu machen.

Beispiel: -13 + 100 = 87

Stellt man nun die Zahl -13 durch ihre Differenz zur Zahl 100, also die 87 dar, so lautet nunmehr die Aufgabe 44 + (-13) in dieser Schreibweise einer negativen Zahl

$$44 + (-13) \mathrel{\hat{=}} 44 + 87 \mathrel{\hat{=}} 131$$

Das Ergebnis ist durch Abziehen der zuvor addierten Dezimalzahl 100 zu korrigieren und lautet dann richtig 31.
Um nun einfach die Differenz zur Dezimalzahl 100 bilden zu können, ergänzt man jede Stelle zur 9 und addiert zum Schluß eine 1.

Beispiel:

$$\begin{array}{r} 99 \\ -13 \\ \hline 86 \\ +\ 1 \\ \hline 87 \end{array}$$

Die Darstellung einer negativen Zahl durch ihre Ergänzung zu einer Basiszahl heißt Komplement. Das Verfahren arbeitet auch, wenn man nicht zur 100, sondern zur 99 ergänzt. Bei dieser Art der Komplementierung spart man die Addition der 1, die man bei der Korrektur wieder hinzufügen muß.

Die gleichen Überlegungen gelten für längere Zahlen mit mehr Stellen und für andere Zahlensysteme. Besonders einfach wird die Komplementbildung im dualen Zahlensystem. Macht man aus jeder 0 eine 1 und aus jeder 1 eine 0, so entspricht das entstandene 1er-Komplement der Ergänzung zur 99 im Dezimalsystem. Addiert man noch eine 1 dazu, so entsteht das bei Mikrocomputern häufig verwendete 2er-Komplement.
Das Beispiel 44 + (-13) lautet mit achtstelligen Dualzahlen:

Zahlenumwandlung

$$44_{10} = 00101100$$

$$13_{10} = 00001101$$

Komplementierung der 13

```
-13   = 11110010   1er-Komplement
      +        1
        11110011   2er-Komplement
```

Addition des 2er-Komplements

```
  00101100
+ 11110011
  100011111
```

Korrektur

```
  100011111
- 100000000
    00011111
```

Zahlenumwandlung

$$00011111_2 = 31_{10}$$

Das Rechenwerk des Mikrocomputers nimmt diese Korrektur automatisch vor. Der Anhang 3 enthält eine Dezimal-Hexadezimal-Tabelle für Dezimalzahlen von -128 bis +127. Der Anhang 4 enthält eine entsprechende Tabelle von 0 bis 1024 nur für positive Zahlen.

Die Rechenregeln für Dualzahlen sind sehr einfach.

Addition	Subtraktion	Multiplikation
0 + 0 = 0	0 - 0 = 0	0 · 0 = 0
0 + 1 = 1	0 - 1 = 1 borgen 1	0 · 1 = 0
1 + 0 = 1	1 - 0 = 1	1 · 0 = 0
1 + 1 = 10	1 - 1 = 0	1 · 1 = 1

Das 1er-Komplement einer Dualzahl macht aus jeder 1 eine 0 und aus jeder 0 eine 1.

Das 2er-Komplement addiert zusätzlich eine 1.

Übungen zum Abschnitt 1.1 und 1.2

1.Aufgabe

Die Dezimalzahl 100 ist nacheinander in eine positive Dualzahl, eine Oktalzahl, eine Hexadezimalzahl und in eine Zahl zur Basis 4 zu verwandeln.
Wie ist die Dezimalzahl -100 als duales 2er-Komplement von acht Stellen darzustellen?

2.Aufgabe

Man führe die folgenden Rechenaufgaben mit achtstelligen Dualzahlen durch:

a.) 47 + 11 =
b.) 100 + (-38) =
c.) 38 +(-100) =
d.) -47 + (-11) =

3.Aufgabe

Ein 8-Bit-Mikrorechner gibt den Inhalt eines Bytes hexadezimal aus

88

a.) Es handelt sich um eine positive Dualzahl, welcher Wert ist es dezimal?
b.) Es handelt sich um eine vorzeichenbehaftete Dualzahl, welcher Wert ist es dezimal?
c.) Es handelt sich um zwei BCD-codierte Dezimalziffern, welcher Wert ist es?

4.Aufgabe

Ein Mikrorechner druckt folgende Bytes hexadezimal aus:

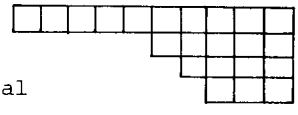
49 43 48 20 48 41 42 53 20 53 41 54 54

Welcher Text im ASCII-Code ist es?

5.Aufgabe

Man verwandle die Dezimalzahl 1000 in Zahlen verschiedener Systeme und trage die Ergebnisse ein.

Dual
Basis 4
Oktal
Hexadezimal



1.3 Rechenschaltungen

Die Digitaltechnik liefert folgende Grundschaltungen:

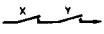
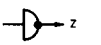
Name	JA-Schaltung	NICHT-Schaltung NOT	UND-Schaltung AND	ODER-Schaltung OR
Kontakte				
Tabelle	X | Z 0 | 0 1 | 1	X | Z 0 | 1 1 | 0	X | Y | Z 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1	X | Y | Z 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1
Symbol				
Formel	Z = X	Z = $\overline{X}$	Z = X · Y	Z = X + Y

Bild 8: Grundschaltungen

Aus diesen Grundschaltungen lassen sich weitere Schaltungen zusammensetzen.

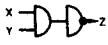
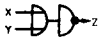
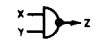
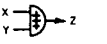
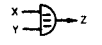
Name	NAND	NOR	Antivalenz XOR	Äquivalenz EQU
Schaltung				
Tabelle	X Y Z 0 0 1 0 1 1 1 0 1 1 1 0	X Y Z 0 0 1 0 1 0 1 0 0 1 1 0	X Y Z 0 0 0 0 1 1 1 0 1 1 1 0	X Y Z 0 0 1 0 1 0 1 0 0 1 1 1
Symbol				
Formel	$Z = \overline{X \cdot Y}$	$Z = \overline{X + Y}$	$Z = (X+Y) \cdot \overline{(X \cdot Y)}$ $Z = X - Y$ $= X \oplus Y$	$Z = (X+Y) \cdot \overline{(X \cdot Y)}$ $Z = \overline{X - Y}$

Bild 9: Zusammengesetzte Schaltungen

Die NICHT-Schaltung bildet das 1er-Komplement entsprechend Bild 10.

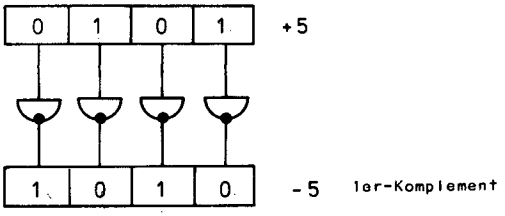
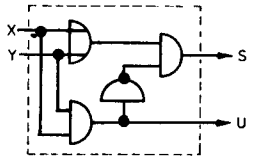


Bild 10: Komplementierschaltung für vier bit

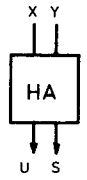
Für das 2er-Komplement ist zusätzlich eine 1 zu addieren.
Für die Addition zweier Dualstellen ist eine Schaltung erforderlich, an deren Eingang die beiden Dualstellen liegen und an deren Ausgang die Summe S und der Übertrag U erscheinen. Dabei sind folgende Kombinationen möglich:

X	Y	U	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Eine XOR-Schaltung liefert die Summe S, eine AND-Schaltung den Übertrag U. Bild 11 zeigt den daraus entwickelten Halbaddierer.



Schaltung



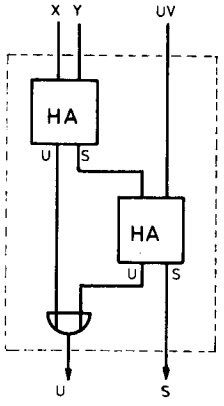
Symbol

Bild 11: Halbaddierer

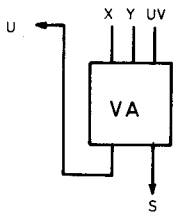
Bei der Addition mehrstelliger Dualzahlen muß der Übertrag UV der vorhergehenden Stelle mit berücksichtigt werden. Dabei sind folgende Kombinationen möglich:

X	Y	UV	U	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Bild 12 zeigt die Schaltung eines Volladdierers.



Schaltung



Symbol

Bild 12: Volladdierer

Der erste Halbaddierer verarbeitet die beiden Dualstellen. Die Summe läuft mit dem Übertrag der vorhergehenden Stelle über einen zweiten Halbaddierer. Eine OR-Schaltung addiert die Überträge der beiden Halbaddierer zum Gesamtübertrag. Bild 13 zeigt ein Addierwerk aus vier Volladdierern, das zwei vierstellige Dualzahlen addieren kann, die parallel auf je vier Leitungen am Eingang ankommen.

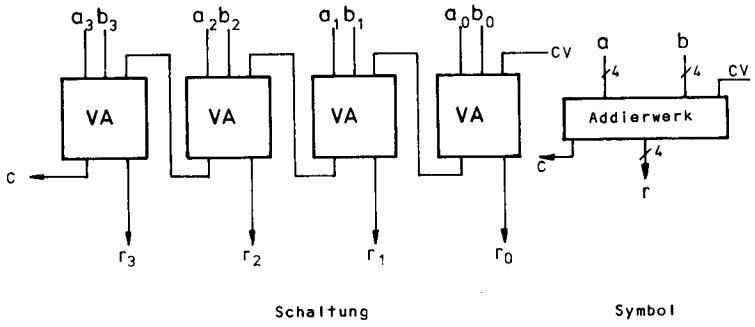


Bild 13: Paralleladdierer

Obwohl zur Addition der beiden wertniedrigsten Stellen nur ein Halbaddierer nötig wäre, enthält die Schaltung an dieser Stelle einen Volladdierer; der Übertrag-Eingang wird mit CV bezeichnet. Dies bedeutet Carry = Übertrag der vorhergehenden Addition, kurz Vorübertrag genannt. Der Ausgang C des werthöchsten Volladdierers führt einen Carry = Übertrag der nächsten Addition zu.

Das Paralleladdierwerk verknüpft zunächst nur zwei 4-Bit-Operanden und einen Vorübertrag CV zu einem 4-Bit-Resultat und einem Übertrag C. Mit einem 4-Bit-Paralleladdierer können folgende Zahlenarten addiert werden:

1. Die Operanden sind positive Dualzahlen von 0 bis +15
Dann bedeutet ein Übertrag C = 1 , daß der zulässige Zahlenbereich überschritten wurde.

Beispiel:

```
        1101      13
       +0101       6   5
C=     10011      19   18
C=1|0010
```

Will man Dualzahlen von mehr als vier bit Länge mit diesem 4-Bit-Paralleladdierer verarbeiten, so muß man in Gruppen von vier Bits von hinten beginnen, den Übertrag am Ausgang des letzten Volladdierers speichern und bei der Addition der nächsten Gruppe als Vorübertrag auf den Eingang CV geben.

Beispiel: Addition zwölfstelliger Dualzahlen

```
3.Gruppe  2.Gruppe  1.Gruppe

   0100      1010      1100  a
 + 0010      0001      0111  b
                          0  CV = 0 gesetzt
                  1 ←— 1 0011  1.Addition gibt C=1
      0 ←— 0 1100              2.Addition gibt C=0
 0 0110                        3.Addition gibt C=0

C=0 0110      1100      0011  Ergebnis
```

Bei der Rechnung mit positiven Dualzahlen bedeutet C=1 (C wie Carry = Übertrag), daß der Zahlenbereich überschritten wurde. Dies wird als Zahlenüberlauf bezeichnet. Bei der gruppenweisen Addition langer Dualzahlen ist der Ausgang C der Übertrag auf die nächste Gruppe.

Ein Übertrag der letzten Gruppe bedeutet auch hier wieder einen Zahlenüberlauf.

2. Die Operanden sind Dualzahlen mit Vorzeichen
Dabei werden negative Zahlen im 2er-Komplement dargestellt.
Der Zahlenbereich erstreckt sich von -8 = 1000 bis +7 = 0111.
Bei der Addition negativer Zahlen kann C = 1 auftreten, ohne daß ein Zahlenüberlauf vorliegt.

Beispiele:

a.)	0011	+3	b.)	0111	+7	c.)	1001	-7
	+0100	+4		+1100	-4		+0100	+4
	0 0111	+7		1 0011	+3		0 1101	-3
d.)	1101	-3	e.)	0111	+7	f.)	1001	-7
	+1100	-4		+0100	+4		+1100	-4
	1 1001	-7		0 1011	11		1 0101	-11

Das C-Bit hat also in dieser Zahlendarstellung keine Bedeutung. Eine Überschreitung des Zahlenbereiches kann in dieser Darstellung nur wie in dem Beispiel e.) bei der Addition zweier positiver Zahlen auftreten, eine Unterschreitung (Unterlauf) nur bei der Addition zweier negativer Zahlen, entsprechend Beispiel f.). Hat das Ergebnis ein anderes Vorzeichen als die beiden Operanden, so liegt es außerhalb des zulässigen Bereiches.

a_3	b_3	r_3	V	
0	0	1	1	Zahlenüberlauf
1	1	0	1	Zahlenunterlauf

Bei allen anderen Kombinationen von a_3 , b_3 und r_3 ist der Überlauf V = oVerflow gleich 0. Die folgende Schaltung Bild 14 prüft diese Bedingung.

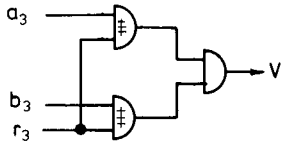


Bild 14: Überlaufbedingung

Das werthöchste Bit des Ergebnisses r_3 zeigt das Vorzeichen an. Es erhält die Bezeichnung N für Negativ.

N = 0 : Ergebnis nicht negativ, also positiv
N = 1 : Ergebnis negativ

Auch in den anderen Zahlen- bzw. Zeichendarstellungen wird das werthöchste Bit als N-Bit bezeichnet, auch wenn es dort kein Vorzeichen darstellt.

3. Die Operanden sind binär codierte Dezimalziffern
Bild 15 zeigt nochmals den BCD-Code zusammen mit den nicht benutzten Kombinationen. BCD bedeutet Binär Codierte Dezimalzahlen

Codierung	Ziffer
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	
1011	
1100	
1101	
1110	
1111	

Bild 15: 8-4-2-1 BCD-Code

Man sieht, daß jede Ziffer durch die entsprechende Dualzahl codiert ist und kann daher den Addierer des Bildes 13 auch zur Addition binär codierter Dezimalzahlen verwenden.

Beispiele:

a.)	0101	5	b.)	0101	5	c.)	0111	7	d.)	1001	9
	0100	4		0101	5		1000	8		1001	9
	01001	9		01010	10		01111	15		10010	18

Die Beispiele b.) und c.) liefern als Ergebnis Bitkombinationen, zu denen keine Dezimalziffer gehört. Das Beispiel d.) zeigt einen Übertrag C=1 und liefert eine gültige Bitkombination, aber ein fehlerhaftes Ergebnis. In allen drei Fällen, in denen sich ein Ergebnis größer 9 oder ein Übertrag C=1 ergibt, erscheint im Dezimalsystem ein Übertrag auf die nächste Stelle. Die Addition von 0110 = 6 korrigiert die entstandene Bitkombination und liefert einen Übertrag C=1, der bei der Verarbeitung weiterer Stellen berücksichtigt werden kann.

Beispiele mit Korrektur:

b.)	0101	5	c.)	0111	7
	+0101	5		+1000	8
	01010			01111	
	+0110	Korrektur		+0110	Korrektur
	C=10000			C=10101	
	Ziffer 0			Ziffer 5	

d.)	1001	9
	+1001	9
	10010	
	+0110	Korrektur
	C=11000	
	Ziffer 8	

Bild 16 zeigt ein 4-Bit-Addierwerk mit nachgeschaltetem Korrekturaddierer, der die Korrekturbedingung prüft und dann die Konstante 0110 dazuaddiert. Bei Mikrocomputern kann man über besondere Befehle entweder die Korrektureinrichtung ein- oder ausschalten oder nach jeder Addition eine Dezimalkorrektur vorneh-

men. Zur Darstellung negativer Dezimalzahlen arbeitet man wie bei Dualzahlen mit dem Komplement.

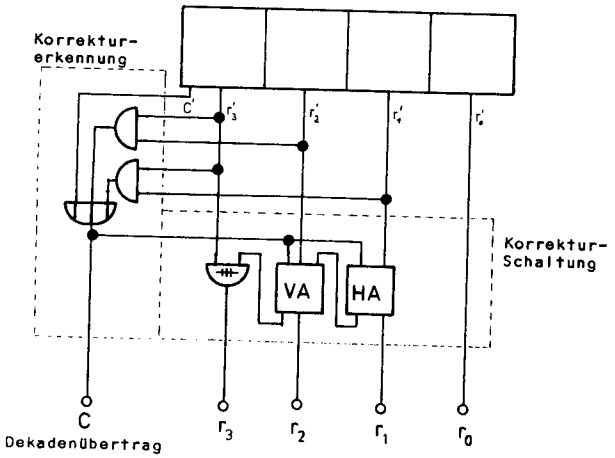


Bild 16: Dezimalkorrektur

4. Die Operanden sind binär codierte Zeichen.
Der 4-Bit-Addierer des Bildes 13 kann nur 16 verschiedene Zeichen wie z.B. die Hexadezimalziffern von 0 bis F verarbeiten. In den häufig verwendeten 8-Bit-Rechenwerken läßt sich jedoch ein Zeichen des ASCII-Codes unterbringen. Mit diesen Zeichen wird man nicht rechnen, sondern nur Vergleiche durchführen. Zwei Zeichen sind dann gleich, wenn ihre Differenz Null ist. Die Schaltung Bild 17 prüft das Ergebnis auf Null.

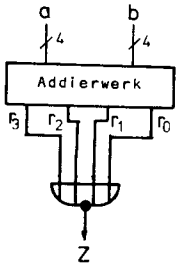


Bild 17: Nullbedingung

Der Ausgang heißt Z - für Zero.

Z = 0 : Das Ergebnis ist nicht Null.
Z = 1 : Das Ergebnis ist Null.

Bei der Subtraktion zweier Dualzahlen mit oder ohne Vorzeichen addiert man das Komplement in einer Schaltung nach Bild 18. Dazu wird der Subtrahend über eine NOT-Schaltung geführt, am Eingang CV liegt konstant eine 1, es wird also das 2er Komplement gebildet.

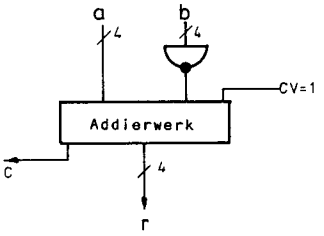


Bild 18: Subtraktionsschaltung

Auch hier muß man wieder die verschiedenen Zahlendarstellungen betrachten.

1. Die Operanden sind positive Dualzahlen von 0 bis +15
Vor jeder Subtraktion sei zunächst CV = 1. Da nur positive Zahlen zugelassen sind, bedeutet ein negatives Ergebnis einen Zahlenunterlauf.

Beispiele:

a.) 4 - (+3) = 4 + (-3)

```
  0100   +4
  1100   -3
     1   CV
C=10001
```

$\overline{C}$=0

b.) 3 - (+4) = 3 + (-4)

```
  0011   +3
  1011   -4
     1   CV
C=01111
```

$\overline{C}$=1

Im Fall a.) wird C=1, obwohl der Zahlenbereich nicht unterschritten wurde. Manche Mikrocomputer negieren deshalb das C-Bit beim Subtrahieren und zeigen dann immer $\overline{C}$=0 an für den Fall, daß das Ergebnis im zulässigen Zahlenbereich liegt. Im Fall b.) wurde der zulässige Zahlenbereich unterschritten, da ein negatives Ergebnis erscheint. Negiert man bei einer Subtraktion das C-Bit, so ergibt sich $\overline{C}$=1 und zeigt damit den Zahlenunterlauf an.

Will man nun Dualzahlen von mehr als vier bit Länge mit dem 4-Bit-Addierer subtrahieren, so muß man wieder in Gruppen von vier Bits von hinten beginnen. Erscheint bei einer Gruppe ein negatives Ergebnis, so muß man von der nächsten Gruppe eine 1 borgen (Borrow), d.h. von dieser Gruppe ist noch zusätzlich eine 1 abzuziehen. Dazu gibt man einfach das ursprüngliche C-Bit bei der nächsten Gruppe auf den Eingang CV.

C = 1 : kein Borgen, addiere eine 1 für 2er Komplement
C = 0 : Borgen, die Addition der 1 unterbleibt.

Beispiel: Subtraktion zwölfstelliger Dualzahlen

```
  0100 1010 0100
- 0010 0001 0111
```

Durch die Negation des zweiten Operanden liegen damit folgende Bitkombinationen am Addierer an:

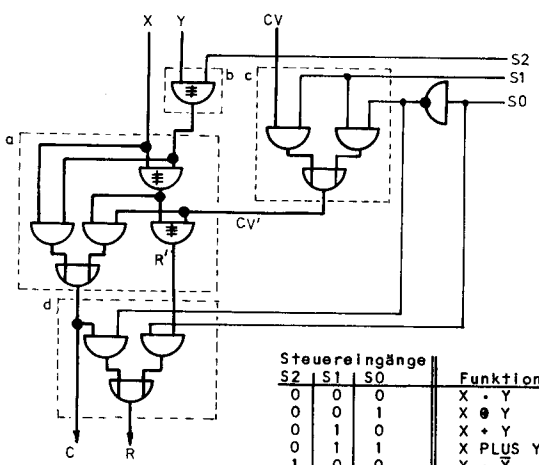
```
3.Gruppe    2.Gruppe    1.Gruppe

   0100        1010        0100   a
 + 1101        1110        1000   b
                              1   C = 1 gesetzt
                   0 ←—— 0 1101   1.Addition gibt C=0
      1 ←—— 1 1000                2.Addition gibt C=1
 1 0010                           3.Addition gibt C=1

C=1 0010        1000        1101   Ergebnis

C̄=0
```

Wird der letzte Übertrag negiert angezeigt, so bedeutet $\overline{C}=0$, daß das Ergebnis im zulässigen Zahlenbereich liegt.

2. Die Operanden sind Dualzahlen mit Vorzeichen.
Der zulässige Zahlenbereich liegt zwischen -8 und +7.
Wie bei der Addition wird man auch hier normalerweise ohne Berücksichtigung der vorhergehenden Operation, also mit CV = 1 arbeiten. Das C-Bit hat auch hier keine Bedeutung, das Ergebnis überschreitet oder unterschreitet nur dann den Zahlenbereich, wenn die Operanden verschiedene Vorzeichen haben. Es liegt die gleiche Überlaufbedingung wie bei der Addition vor, wenn man das negierte Vorzeichen des Subtrahenden berücksichtigt.

3. Die Operanden sind binär codierte Dezimalziffern.
Auch hier läßt sich die Subtraktion auf die Addition des Komplementes zurückführen.

1.4 Die Arithmetisch-Logische Einheit (ALU)



Steuereingänge S2	S1	S0	Funktion
0	0	0	X · Y
0	0	1	X ⊕ Y
0	1	0	X + Y
0	1	1	X PLUS Y
1	0	0	X · $\overline{Y}$
1	0	1	X ⊕ $\overline{Y}$
1	1	0	X + $\overline{Y}$
1	1	1	X PLUS $\overline{Y}$ = X MINUS Y

Bild 19: Schaltung eines ALU-Elementes

Eine Erweiterung des Volladdierers bietet die Möglichkeit, neben der Addition auch die Subtraktion und die logischen Funktionen AND, OR und XOk auszuführen. Das ALU-Element Bild 19 enthält als Grundschaltung einen gegenüber Bild 12 vereinfachten Volladdierer (Schaltungsteil a). Schaltungsteil b negiert den Operanden Y und ermöglicht somit die Subtraktion. Schaltungsteil c übernimmt den Vorübertrag CV unverändert (arithmetische Operationen) oder setzt ihn konstant 0 bzw. 1 (logische Operationen). Schaltungsteil d übernimmt entweder C' (logische Operationen) oder R' (arithmetische Operationen) am Ergebnisausgang R. Die Steuereingänge S0, S1 und S2 wählen die Funktion aus.

Bild 20 zeigt das Schema einer arithmetisch-logischen Einheit für ein 4-Bit-Wort, die aus vier Elementen des Bildes 19 aufgebaut ist. Eine Zusatzschaltung untersucht das Resultat auf Übertrag C, Überlauf V, Negativ N und Zero (Null) Z. Diese Bedingungen werden später für Sprünge und Verzweigungen benötigt.

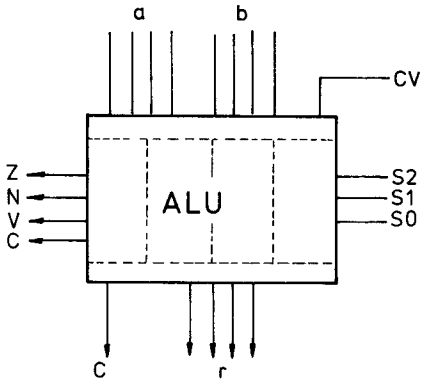
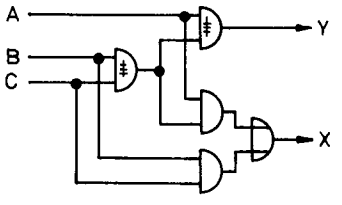


Bild 20: Arithmetisch-logische Einheit für vier bit

Übungen zum Abschnitt 1.3 und 1.4

1.Aufgabe
Man stelle für die folgende Schaltung die Wertetabelle für alle Kombinationen der Eingangsgrößen auf.



A	B	C	X	Y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

2.Aufgabe
Am Eingang einer 4-Bit-ALU Bild 20 liegen die Bitkombinationen A = 0011 und B = 0101. Wie lauten das Ergebnis und die vier Bedingungen für folgende Operationen:

a.) Addition mit CV = 0
b.) Subtraktion für CV = 1
c.) Logisches UND (AND)
d.) Logisches ODER (OR)
e.) Logisches exklusives ODER (XOR)

3.Aufgabe
An einer 8-Bit-Arithmetisch-logischen Einheit wird die Funktion ADDIERE ausgewählt. Die beiden Operanden sind

Operand A	Operand B	CV
10010000	00000111	1

a. Man gebe das Ergebnis binär und hexadezimal an.

b. Das Ergebnis sei eine Dualzahl mit Vorzeichen im 2er Komplement. Wie lautet sie dezimal?

c. Das Ergebnis sei eine positive Dualzahl.
Wie lautet die entsprechende Dezimalzahl?

d. Das Ergebnis sei eine zweistellige BCD-codierte Dezimalzahl.

1.5 Speicherschaltungen

Speicherschaltungen haben die Aufgabe, binäre Zustände (0 bzw.1) aufzunehmen, zu speichern und wieder abzugeben.

Die einfachste Speicherschaltung besteht aus zwei rückgekoppelten NAND-Schaltungen nach Bild 21.

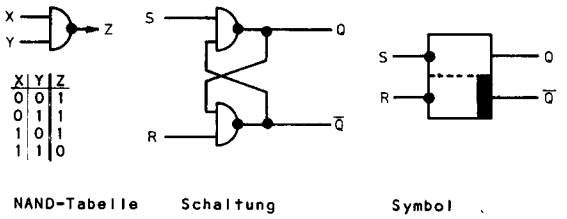


Bild 21: NAND-Basis-Flip-Flop

An den beiden Eingängen S und R können vier verschiedene Kombinationen anliegen. Die beiden Ausgänge Q und $\overline{Q}$ sollen komplementär sein.

a. S = 1 und R = 1

Ein Blick auf die Wertetabelle der NAND-Schaltung zeigt, daß für diesen Fall die Ausgänge der NANDs vom Zustand der rückgeführten zweiten Eingänge abhängen. Nur die beiden Zustände Q = 0 und $\overline{Q}$ = 1 sowie Q = 1 und $\overline{Q}$ = 0 nach Bild 22 genügen der NAND-Bedingung und sind stabil. Die Eingänge R = S = 1 bewahren

also den Zustand Q der Schaltung auf, der Ausgang $\overline{Q}$ ist die Verneinung oder das Komplement des Ausgangs Q.

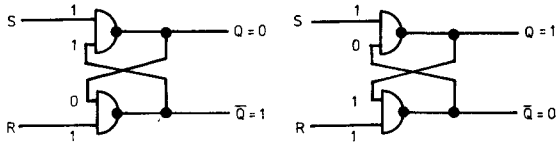


Bild 22: Stabile Speicherzustände des NAND-Basis-Flip-Flops

b. S = 1 und R = 0

Der Eingang R = 0 erzwingt am Ausgang $\overline{Q}$ des unteren NANDs eine 1, durch die Rückführung auf das obere NAND und S = 1 ergibt sich Q = 0. Unabhängig vom vorhergehenden Zustand wird die Schaltung auf Q = 0 rückgesetzt.

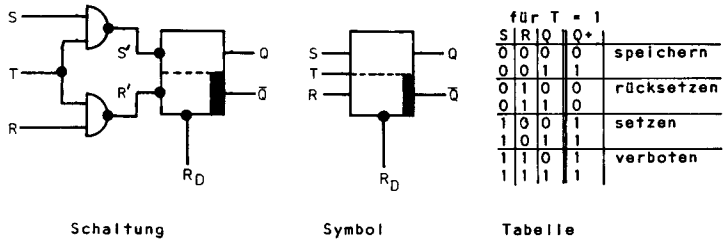
c. S = 0 und R = 1

Der Eingang S = 0 erzwingt am Ausgang Q des oberen NANDs eine 1, durch die Rückführung auf das untere NAND und R = 1 ergibt sich $\overline{Q}$ = 0. Unabhängig vom vorhergehenden Zustand wird die Schaltung auf Q = 1 gesetzt.

d. S = 0 und R = 0

Entsprechend der Wertetabelle der NAND-Schaltung führt diese Kombination auf die Ausgänge Q = $\overline{Q}$ = 1 ; dies ist jedoch ein Widerspruch zu der Forderung, daß beide Ausgänge komplementär sein sollen.

Bild 23 zeigt eine getaktete Speicherschaltung, bei der eine zusätzliche Eingangsschaltung als Tor wirkt und gleichzeitig die Eingangssignale negiert.



für T = 1

S	R	Q	Q+	
0	0	0	0	speichern
0	0	1	1	
0	1	0	0	rücksetzen
0	1	1	0	
1	0	0	1	setzen
1	0	1	1	
1	1	0	1	verboten
1	1	1	1	

Tabelle

Bild 23: Taktzustandsgesteuertes RS-Flip-Flop

Der Eingang R_D setzt taktunabhängig $\overline{Q}$=1 und damit Q=0 .

Für T = 0 liefert die Eingangsschaltung immer die Werte S = R = 1, das nachgeschaltete Basis-Flip-Flop bleibt unverändert.
Für T = 1 negiert jetzt die Eingangsschaltung die ankommenden Signale. Der neue Zustand Q+ hängt von den Eingängen S und R sowie vom alten Zustand Q ab, ist also eine Funktion von drei Variablen entsprechend Bild 23.

Für R = S = 0 bleibt der alte Zustand erhalten.
Für R = 1 wird das Flip-Flop auf Q = 0 rückgesetzt.
Für S = 1 wird das Flip-Flop auf Q = 1 gesetzt.
Der Fall R = S = 1 bringt zunächst beide Ausgänge Q und $\overline{Q}$ auf 1; ändert sich jedoch der Takt von 1 auf 0, so geht die Schaltung wieder in einen stabilen Zustand über, bei dem Q und $\overline{Q}$ komplementär sind. Der neue Speicherinhalt wird durch das Verhalten der Schaltelemente bestimmt und ist nicht mehr logisch vorhersehbar. Eine einfache Eingangsschaltung nach Bild 24 verhindert dies und leitet ein auf der Leitung ankommendes Signal in den Speicher.

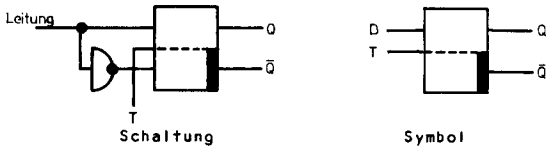


Bild 24: D-Flip-Flop

Nehmen die bisher gezeigten Speicher an ihren Eingängen ein Signal auf, so wird der alte Speicherinhalt sofort zerstört. Für viele Anwendungen ist es jedoch notwendig, am Eingang ein neues Signal aufzunehmen, am Ausgang den alten Zustand aber noch beizubehalten. Aus diesem Grund führt man einen Zwischenspeicher ein.

Die Schaltung Bild 25 besteht aus zwei taktzustandsgesteuerten RS-Flip-Flops. Dabei wird dem Slave der negierte Takt zugeführt. Ist der Master geöffnet, so ist der Slave gesperrt und umgekehrt.

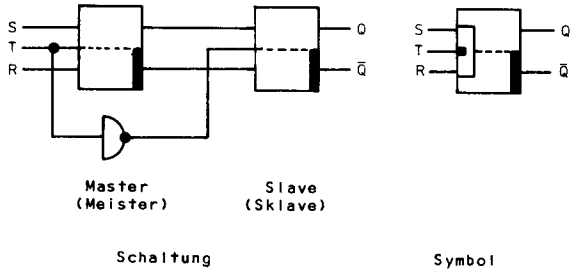


Bild 25: RS-Master-Slave-Flip-Flop

Bild 26 zeigt, wie die Schaltung für T = 1 im Master den neuen Inhalt und im Slave den alten Zustand speichert.

Neben diesem taktzustandsgesteuerten Flip-Flop gibt es auch Speicher, bei denen die Flanke des Taktes (Wechsel von 0 auf 1 bzw. 1 auf 0) die Übernahme der Signale bewirkt.

Zur Speicherung des C-Bits dient ein Master-Slave-Flip-Flop, das den Vorübertrag an die arithmetisch-logische Einheit abgibt und gleichzeitig den neuen Übertrag aufnimmt.

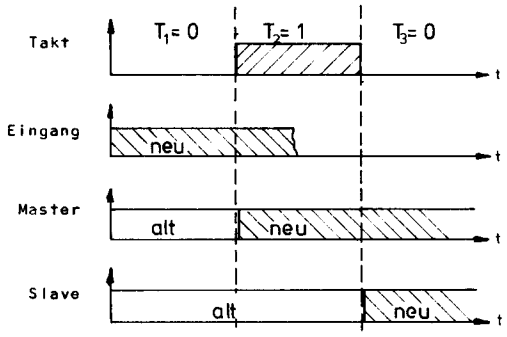


Bild 26: Datenübergabe beim Master-Slave-Flip-Flop

Das parallele Schieberegister Bild 27 nimmt in einem Takt vier bit an den vier Eingängen auf. Für den Datentransport sind vier Leitungen erforderlich.

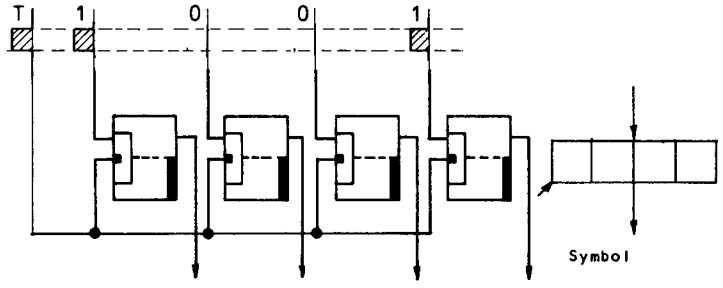


Bild 27: Paralleles Schieberegister für vier bit

Verbindet man den Ausgang eines Flip-Flops mit dem Eingang des rechts folgenden, so entsteht ein Rechts-Schieberegister nach Bild 28. Der ursprüngliche Inhalt sei 1001. Legt man am Eingang E eine 0 an und gibt einen Impuls auf die Taktleitungen, so rükken alle Bits um eine Position nach rechts. Der neue Inhalt ist 0100.

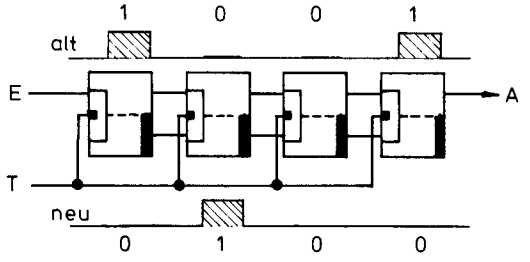


Bild 28: Rechts-Schieberegister für vier bit

Über zusätzliche Schaltungen lassen sich in Schieberegistern folgende Operationen mit dem Registerinhalt durchführen:

a. Wahlweise nach rechts oder links schieben.
b. Frei werdende Stellen mit 0 füllen (logisches Schieben).
c. Frei werdende Stellen beim Schieben nach rechts mit dem Vorzeichenbit füllen (algebraisches Schieben).
d. Den Ausgang A auf den Eingang E schalten (zyklisches Schieben oder Rotieren).

Bild 29 zeigt das ~~akkumulative (sammelnde)~~ Register des Rechenwerkes. Dieser Akkumulator gibt an seinem Ausgang einen Operanden an die arithmetisch-logische Einheit ab und nimmt gleichzeitig an seinem Eingang das Ergebnis auf; der zweite Operand kommt aus dem Datenspeicher. Ein fünftes Flip-Flop nimmt den Übertrag C auf. Schickt man den Inhalt des Akkumulators allein über die arithmetisch-logische Einheit, so können dort durch Zusatzschaltungen auch Schiebeoperationen ausgeführt werden.

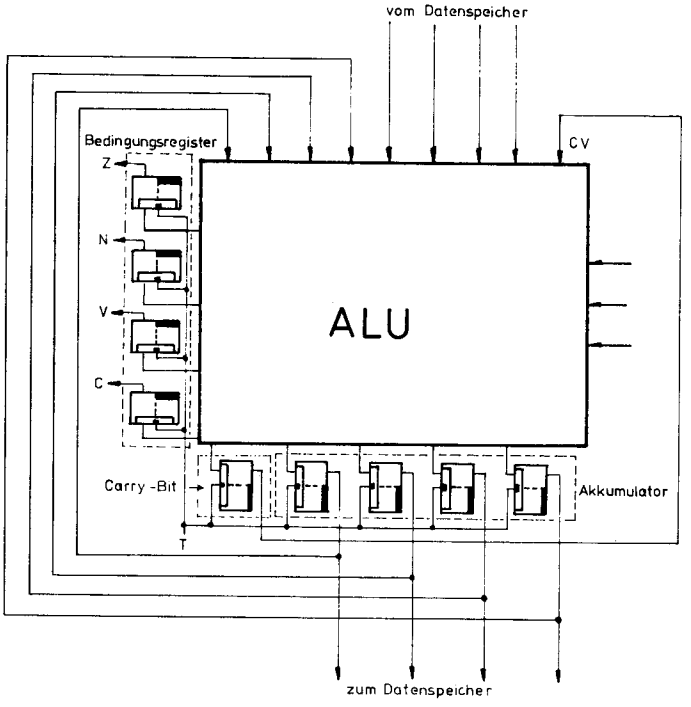


Bild 29: Akkumulator, Bedingungsregister und Carry-Bit

Eine nicht eingezeichnete Prüfschaltung untersucht den Ausgang der arithmetisch-logischen Einheit auf Null (Z-Bit), Vorzeichen (N-Bit), Überlauf (V-Bit) und Übertrag (C-Bit). Das Bedingungsregister speichert die vier Bits.

Das binäre Zählelement nach Bild 30 entsteht durch Rückführung der Ausgänge Q und $\overline{Q}$ auf die Eingänge R und S.

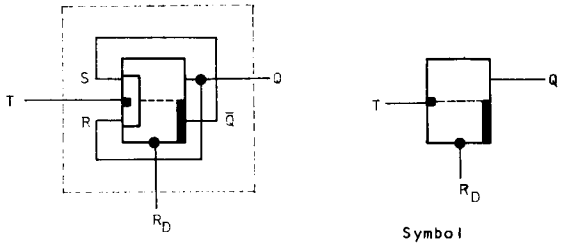


Bild 30: Zählelement

Durch Überkreuzung der Leitungen (Q nach R und $\overline{Q}$ nach S) schaltet das Flip-Flop mit der halben Taktfrequenz dauernd um. Bild 31 zeigt den zeitlichen Verlauf des Eingangstaktes T und der Ausgänge Q und $\overline{Q}$. Die Taktfrequenz erscheint halbiert am Ausgang.

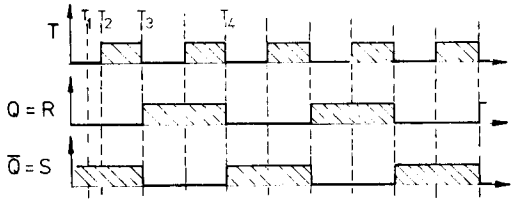


Bild 31: Arbeitsweise eines Zählelementes

Bild 32 zeigt eine Hintereinanderschaltung von drei Zählelementen zu einem 4-Bit-Zähler, der sich über die taktunabhängigen Eingänge auf einen Anfangswert einstellen läßt. Sieht man die

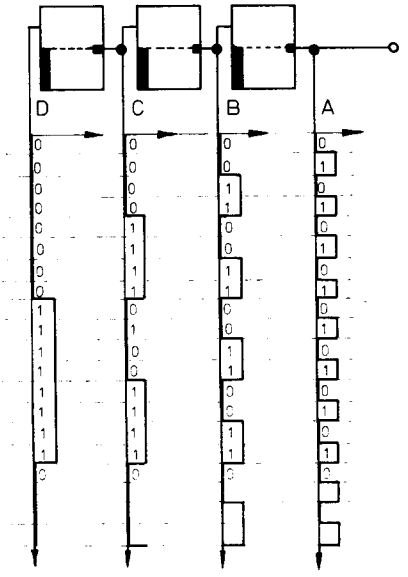


Bild 32:
Vierstelliger
Dualzähler

Signale auf den Leitungen A,B,C und D als Dualzahlen an, so erscheinen die Zahlen von 0000 bis 1111. Da jedes Flip-Flop das folgende ansteuert, schalten die einzelnen Stufen nicht gleichzeitig (asynchron). Durch einen einzigen Impuls auf die Taktleitung kann der Inhalt des Registers um 1 erhöht werden, ohne daß ein Addierwerk benötigt wird.

Taktunabhängige Eingänge ermöglichen das Laden des Zählers mit einem Anfangswert.
Über zusätzliche Schaltungen läßt sich der Inhalt des Registers durch einen Taktimpuls um 1 vermindern.

Die bisher behandelten Schaltungen dienten zum Speichern und Verarbeiten von Daten. Für eine automatische Ausführung von Rechen- und Transportoperationen werden Schaltwerke benötigt, die bestimmte Signalfolgen aussenden und damit nacheinander Leitungswege durchschalten.

Ausgehend von einem seriellen Schieberegister Bild 28 zeigt Bild 33 einen Ringzähler. Das START-Signal setzt das Register auf den Ausgangszustand 1000. Die Taktimpulse lassen die 1 umlaufen, da der Ausgang auf den Eingang geschaltet ist. Alle Stufen schalten gleichzeitig (synchron). Nach einem Taktimpuls ist der neue Zustand 0100.

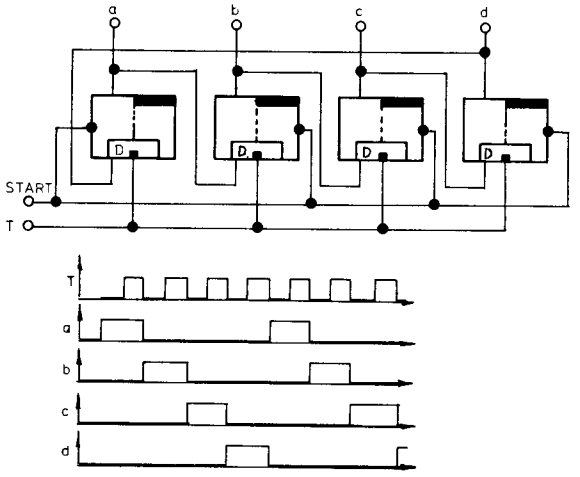


Bild 33: Ringzähler

Der Ringzähler sendet immer die gleiche Folge von Steuersignalen aus. Das Umschalten zwischen den Flip-Flops kann durch äußere Bedingungen beeinflußt werden. Bild 34 zeigt ein Schaltwerk, das über die Steuereingänge S1 und S2 geschaltet wird.

Die Steuereingänge bestimmen, welche der drei möglichen Signalfolgen auf den Leitungen a,b,c und d ausgesendet wird. Die Signalleitungen schalten nacheinander Datenleitungen oder Takteingänge von Speichern und ermöglichen damit einen automatischen Ablauf von Transport- und Rechenvorgängen.

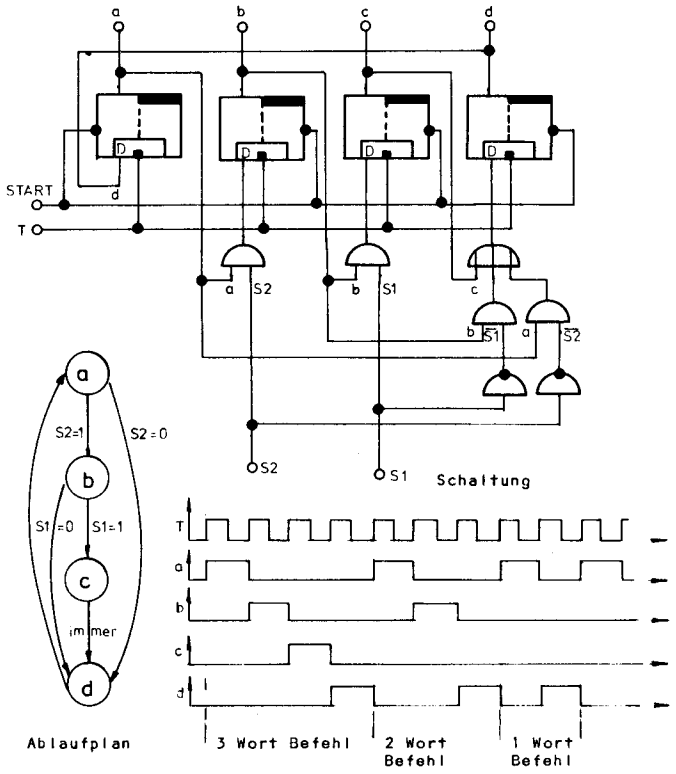


Bild 34: Befehls-Ablaufsteuerung

Die Schaltung ist ein Beispiel für eine Ablaufsteuerung zum Holen von Befehlen, die aus 1 oder 2 oder 3 Wörtern bestehen.

Zustand a: immer 1. Wort mit Funktionscode holen.
Zustand b: für S2=1 2. Wort mit Adresse oder Konstanten holen.
Zustand c: für S1=1 3. Wort mit Adresse holen.
Zustand d: immer den Befehl ausführen.

Die beiden Steuergrößen S1 und S2 sind Bestandteil des Funktionscodes und enthalten die Länge des Befehls. Die Ablaufsteuerung ermöglicht den automatischen Ablauf einer Befehlsfolge (Programm). Für die eigentliche Befehlsausführung ist ein weiteres Schaltwerk erforderlich.

Eine Ablaufsteuerung kann also verschiedene Befehle ausführen, genauso wie die arithmetisch-logische Einheit verschiedene Operationen mit Daten durchführt. In beiden Fällen wählen die Steuereingänge die gewünschte Operation aus. Speichert man also die Bitkombinationen für die Steuereingänge in einem Befehlsspeicher und führt sie nacheinander dem Befehls-Schaltwerk und der arithmetisch-logischen Einheit zu, so lassen sich die vorgesehenen Funktionen automatisch ausführen.

1.6 Datenspeicher

Datenspeicher dienen zur Aufbewahrung großer Datenmengen über längere Zeit, während der Akkumulator nur kurzzeitig Zwischenergebnisse aufnehmen kann. Die Daten werden zu mehreren bit, z. B. vier, zu einem Speicherwort zusammengefaßt, das durch die Vergabe einer Nummer oder Adresse eindeutig bestimmt wird. Bild 35 zeigt eine Anordnung von acht Wörtern zu je vier bit in einem Nur-Lese-Speicher (ROM).

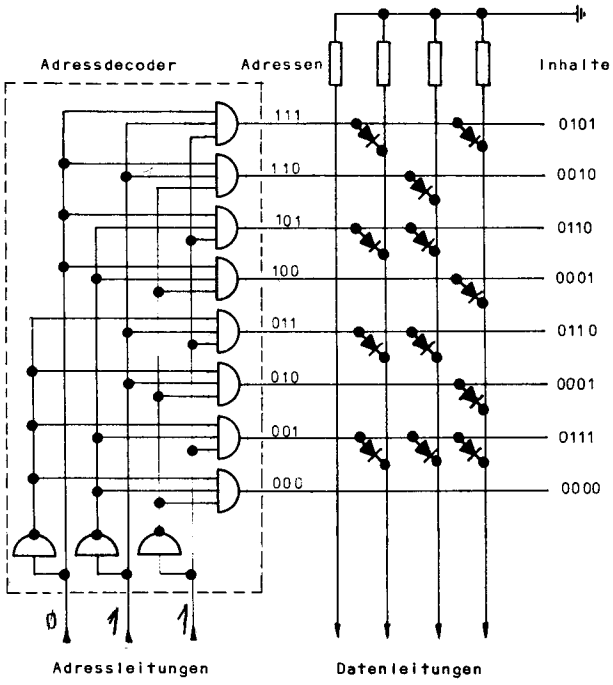


Bild 35: ROM-Speicher für 8 Wörter

Die Adreßleitungen wählen über einen Adreßdecoder eins von acht Wörtern aus, die Datenleitungen führen die Inhalte der ausgewählten Speicherstellen dem Rechenwerk zu.

Beim Matrixspeicher Bild 36 werden nur die Speicherstellen angesprochen, die im Kreuzungspunkt einer ausgewählten Zeilen- und Spaltenleitung liegen. Das Beispiel zeigt eine Anordnung von 16 Wörtern zu je vier bit, von denen jedoch nur ein Bit dargestellt werden konnte, die restlichen drei liegen hinter der Zeichenebene.

Die hier dargestellten Speicher erlauben einen wahlfreien Speicherzugriff, Random Access genannt: jedes Speicherwort ist durch die Angabe einer Adresse erreichbar.
Bei den Schreib-Lese-Speichern (RAM = Random Access Memory) bestehen die Speicherelemente je nach Herstellungsart aus Flip-Flops, Transistoren oder einfach Kapazitäten. Zur Auswahl der

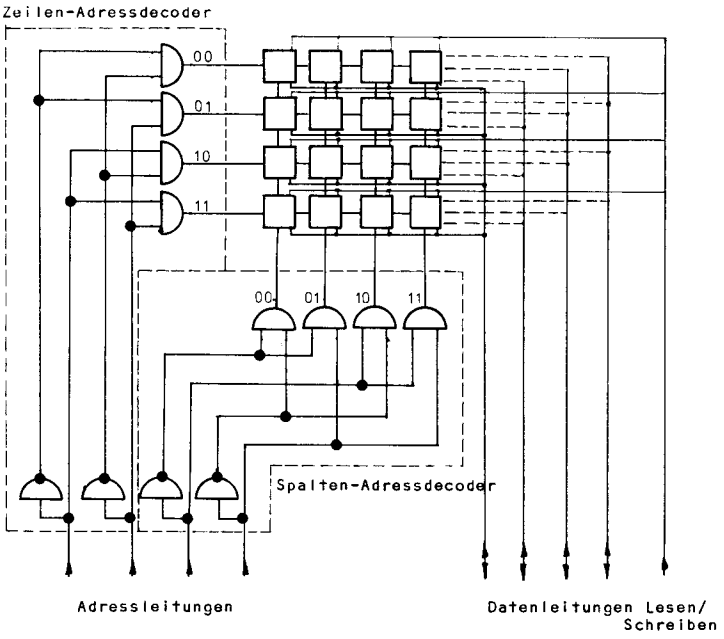


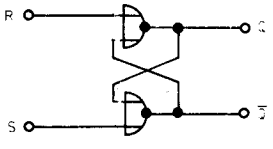
Bild 36: Matrixspeicher für 16 Wörter

Schreib- bzw. Lesefunktion ist eine Auswahlleitung Schreiben/ Lesen erforderlich. Die Datenleitungen werden in beiden Richtungen betrieben.

Bei den Nur-Lese-Speichern oder Festwertspeichern (ROM = Read Only Memory) bestehen die Speicherelemente aus Dioden oder Widerständen, die beim Herstellprozeß oder später vom Benutzer (PROM = Programmable ROM) so behandelt werden, daß sie entweder eine 0 oder eine 1 enthalten. Nach dieser Festlegung kann der Speicherinhalt nicht mehr verändert werden, ein Schreiben ist nicht möglich. Bei den löschbaren Festwertspeichern (EPROM = Erasable PROM) bestehen die Speicherelemente aus elektrischen Ladungen; der gesamte Speicherinhalt kann mit UV-Licht gelöscht und später elektrisch neu programmiert werden.

Übungen zum Abschnitt 1.5 und 1.6

1.Aufgabe
Ähnlich wie aus zwei NANDs läßt sich auch aus zwei rückgekoppelten NOR-Schaltungen eine Speicherschaltung aufbauen. Für das NOR-Basis-Flip-Flop stelle man die Wertetabelle auf.
Q ist der alte, Q+ der neue Zustand.



NOR- Basis-Flip-Flop

S	R	Q	Q+
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2.Aufgabe
Der Inhalt eines seriellen 4-Bit-Schieberegisters ähnlich Bild 28 sei 1011. Er wird verschiedenen Schiebeoperationen unterworfen. Man gebe den neuen Inhalt des Registers an.

a.) Verschiebung um ein Bit nach rechts, am Eingang E liegt konstant eine 0.
b.) Verschiebung um ein Bit nach links, am Eingang des letzten Flip-Flops liegt konstant eine 0.
c.) Verschiebung um ein Bit nach rechts, der Ausgang A wird auf den Eingang E rückgeführt.
d.) Verschiebung um ein Bit nach links, der Ausgang des ersten Flip-Flops wird auf den Eingang des letzten rückgeführt.
e.) Verschiebung um ein Bit nach rechts, der Inhalt des ersten Flip-Flops liegt am Eingang E.

3.Aufgabe
Auf den Adreßleitungen des Speichers Bild 35 liegt die Adresse 011 an. Welcher Speicherinhalt wird gelesen?

1.7 Ein einfaches Rechenwerk mit Datenspeicher und Ein-/Ausgaberegistern

Ausgehend vom Bild 29 (arithmetisch-logische Einheit mit Akkumulator, Bedingungsregister und Übertragspeicher) zeigt Bild 37 ein einfaches Modell-Rechenwerk. Ein Datenspeicher nimmt 128 Speicherstellen zu je vier bit auf. Das Adressregister dient zur Auswahl der $2^7 = 128$ Adressen. Die Speicheradresse 0000 0000

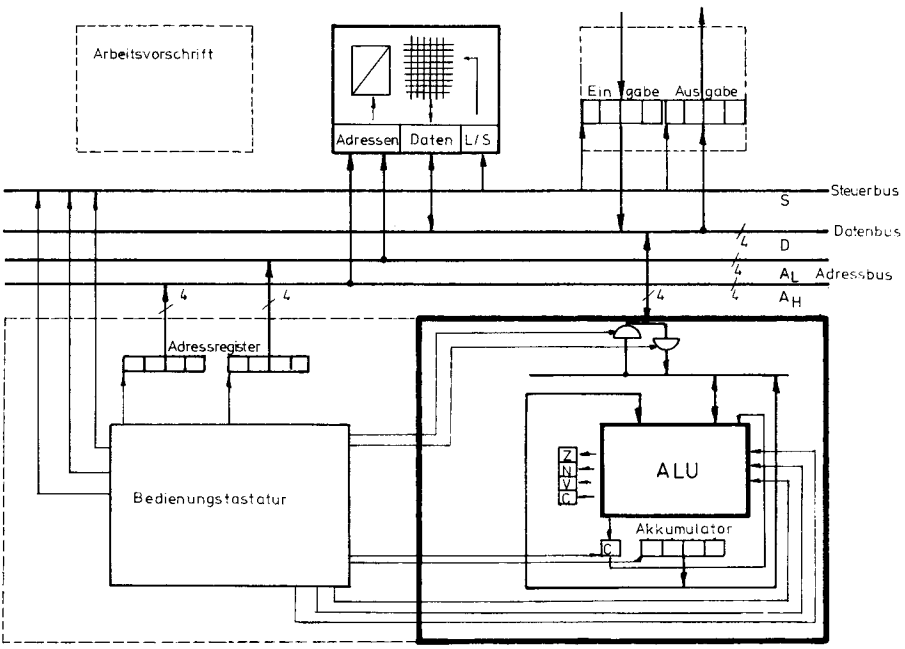
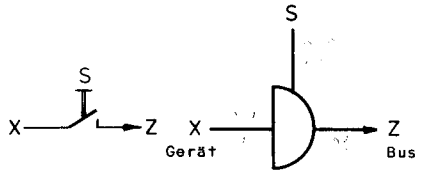


Bild 37: Rechenwerk mit Datenspeicher und Ein/Ausgabe-Registern

enthält ein besonderes Register zur Dateneingabe, die Adresse 0000 0001 ein Register zur Datenausgabe.

Die Adreß- und Datenleitungen aller Bauteile (arithmetisch-logische Einheit, Speicher, Register) führen auf ein gemeinsames Leitungsbündel, Bus genannt. Der Datenbus transportiert ein 4-Bit-Datenwort parallel auf vier Leitungen, der Adreßbus nimmt eine 8-Bit-Adresse auf acht Leitungen auf. Während der Adreßbus nur in einer Richtung vom Adreßregister zum Speicher betrieben wird, arbeitet der Datenbus in beiden Richtungen (bidirektional). Geben mehrere Geräte ihre logischen Signale 0 und 1 auf eine gemeinsame Leitung, so beeinflussen sie sich gegenseitig. Zur Entkopplung werden alle Anschlüsse mit Tri-State-Toren versehen, die neben den Zuständen 0 und 1 den dritten Zustand "hochohmig" haben und damit das Bauelement vom Bus trennen.
Bild 38 zeigt ein solches Tor mit Wertetabelle.



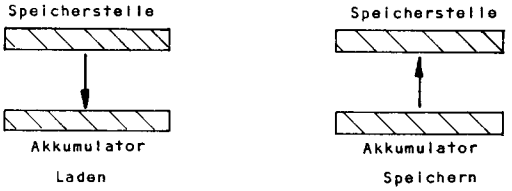
S	X	Z
0	0	hochohmig
0	1	
1	0	0
1	1	1

Bild 38: Tri-State-Tor

Für den Steuereingang S = 0 ist der Ausgang Z hochohmig und trennt das Gerät von der Leitung. Für S = 1 schaltet das Tor den Eingang X auf den Ausgang Z.
Mit dem Rechnermodell des Bildes 37 sind folgende Operationen möglich:

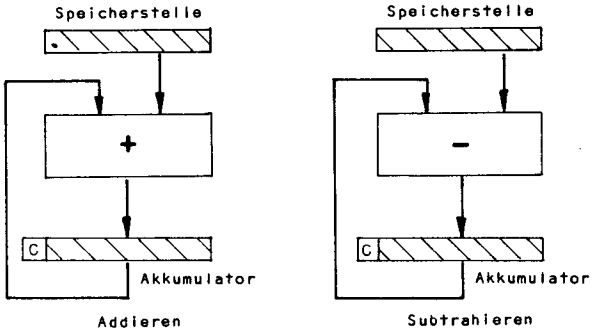
1. Datentransport

Laden des Akkumulators mit dem Inhalt einer Speicherstelle bzw. eines Eingaberegisters oder Speichern des Akkumulatorinhaltes in eine Speicherstelle bzw. in das Ausgaberegister. Der Inhalt des abgebenden Speichers bleibt erhalten, der Inhalt der aufnehmenden Speicherstelle wird überschrieben.



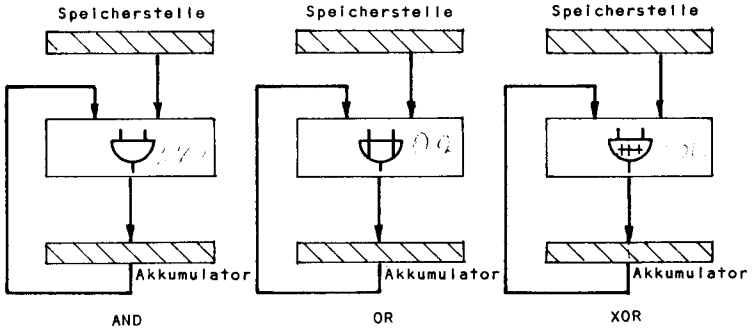
2. Arithmetische Operationen

Addition und Subtraktion des Akkumulatorinhaltes mit dem Inhalt einer Speicherstelle, der alte Akkumulatorinhalt wird durch das Ergebnis überschrieben. Das Ergebnis beeinflußt das Bedingungsregister. Der neue Übertrag überschreibt den alten Inhalt des C-Bits.



3. Logische Operationen

Die Operationen AND, OR und XOR verknüpfen den Inhalt des Akkumulators mit dem Inhalt einer Speicherstelle. Das Ergebnis erscheint wieder im Akkumulator und beeinflußt das Bedingungsregister. Das C-Bit bleibt unverändert.



4. Schiebeoperationen

Die Operation Schiebe logisch rechts schiebt den Akkumulatorinhalt um ein Bit nach rechts und füllt das werthöchste Bit mit einer 0. Die Operation Schiebe zyklisch rechts schiebt den Akkumulatorinhalt um ein Bit nach rechts und setzt das wertniedrigste Bit an die Stelle des werthöchsten. Das C-Bit bleibt unverändert, das Bedingungsregister wird beeinflußt.

Die Bedienung des Modell-Rechenwerkes ist umständlich. Jede Operation muß in mehreren Schritten mit einer Tastatur ausgeführt werden. Für eine Addition des Akkumulatorinhaltes mit dem Inhalt der Speicherstelle 13 wäre folgendes zu tun:

- Adresse 0000 1101 am Adressregister einstellen
- Signal "Lesen" am Datenspeicher anlegen

- Funktion "Addieren" an den Steuereingängen der ALU
- Datenweg Speicher - ALU durchschalten
- Akkumulatorinhalt auf die ALU schalten
- Ergebnis in den Akkumulator übernehmen

1.8 Ein einfaches Steuerwerk mit Programmspeicher

Die Bedienungstastatur des Bildes 37 verlangt vom Benutzer eine binäre Tastenbetätigung (drücken oder nicht drücken) in einer bestimmten Reihenfolge. Zum automatischen Ablauf einer Operation muß man die nötigen Schalterstellungen zusammen mit den Datenadressen in einem Programmspeicher ablegen. An die Stelle des Bedienungspultes tritt ein Schaltwerk, das aufgrund der ihm übergebenen Bitkombinationen - Befehle genannt - den Arbeitsablauf steuert. Bild 39 zeigt den Aufbau eines Steuerwerkes mit Programmspeicher. Das Programm besteht aus unveränderlichen Daten (Konstanten) und Befehlen. Es liegt in einem Nur-Lese-Speicher (ROM), der 128 Speicherstellen zu je vier bit aufnehmen kann. Zur Codierung der Funktion verwendet das Modell ein Wort von vier bit, damit sind 16 verschiedene Befehlsarten möglich.

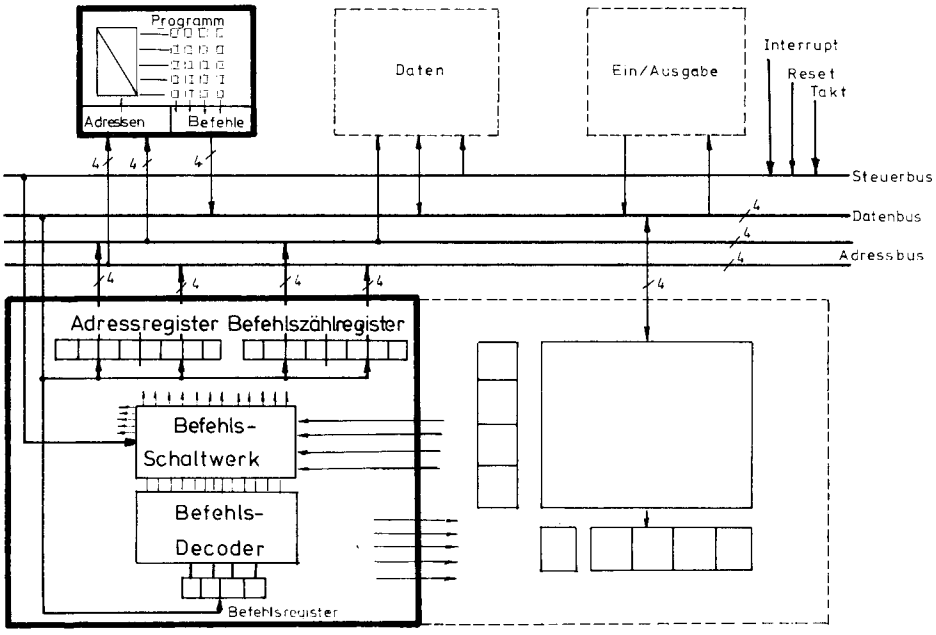


Bild 39: Steuerwerk und Programmspeicher

Die Befehle werden wie die Daten über die Adreßleitungen angesprochen. Das Befehlszählregister BZR enthält die Adresse des als nächsten auszuführenden Befehls. Es besteht aus einem einstellbaren Dualzähler für acht bit mit folgenden Funktionen:

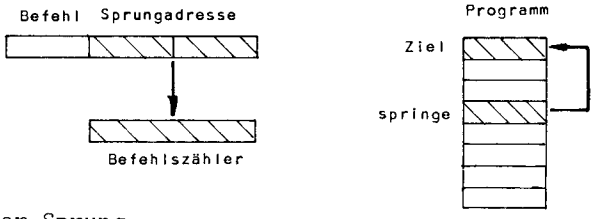
a. Einstellen eines bestimmten festen Anfangswertes beim Starten des Steuerwerkes (Reset) oder beim Betätigen einer äußeren Steuerleitung zur Unterbrechung des Programms (Interrupt).
b. Erhöhung des Zählerstandes um 1, wenn ein 4-Bit-Befehlswort aus dem Speicher geholt wurde. In dieser Betriebsart werden die Befehle in der Reihenfolge ausgeführt, in der sie im Programmspeicher liegen.
c. Laden des Zählers mit einer neuen Adresse, die über den Datenbus aus dem Programmspeicher kommt. Dadurch kann das Programm an einer anderen Stelle fortgesetzt werden. Diese Sprungbefehle enthalten neben dem 4-Bit-Funktionscode die Adresse des nächsten Befehls (Sprungziel) von acht bit Länge.

Das Befehlsregister nimmt den Funktionscode, das Adressregister die Datenadressen auf. Der Befehlsdecoder besteht aus einem Festwertspeicher, der den 4-Bit-Funktionscode als Adresse aufnimmt und ein 12-Bit-Wort zur Steuerung des Befehlssteuerwerkes abgibt. Das Befehlsschaltwerk nimmt den decodierten Funktionscode, die Anzeigen des Bedingungsregisters und die äußeren Signale Restart und Interrupt auf und gibt die Steuersignale an die Register, die arithmetisch-logische Einheit, die Tri-State-Tore und die Speicher ab.

Zu den Rechenwerksbefehlen des Abschnitts 1.7. kommen folgende Sprungbefehle, die im ersten Wort den Funktionscode und im zweiten und dritten Wort die Sprungadresse enthalten:

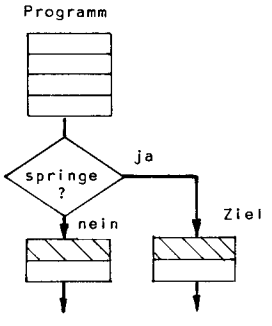
1. Unbedingter Sprung

Der Adreßteil des Befehls überschreibt bedingungslos den Inhalt des Befehlszählregisters.



2. Bedingter Sprung

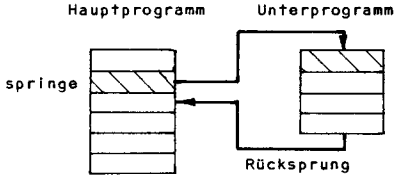
Das Schaltwerk untersucht das Bedingungsregister des Rechenwerks. Das Modell sieht die Befehle Springe bei positiv (N-Bit = 0), Springe bei negativ (N-Bit = 1) und Springe bei Null (Z-Bit = 1) vor. Ist die Bedingung erfüllt, so ersetzt der Adreßteil den



Inhalt des Befehlszählers (Sprung); ist sie nicht erfüllt, so zählt das Register um 1 weiter und der nächste Befehl wird ausgeführt.

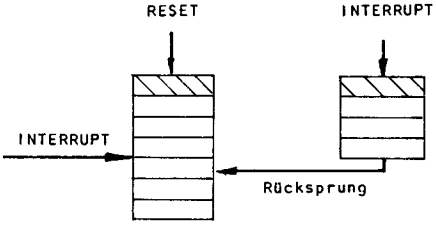
3. Unterprogrammsprung

Der Befehl Unterprogrammsprung rettet den alten Befehlszählerstand in eine Datenspeicherstelle und lädt dann erst die Adresse des Sprungziels. Durch den Befehl Rücksprung am Ende des gerufenen Unterprogramms wird der alte Wert des Befehlszählers zurückgeladen und das Hauptprogramm setzt seine Arbeit fort.



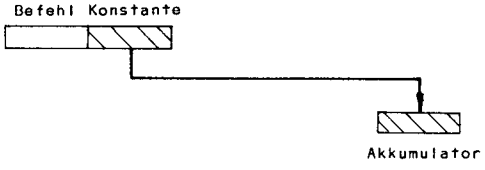
4. Äußere Steuersignale

Zu Beginn der Programmausführung muß das Steuerwerk in einen bestimmten Anfangszustand gebracht werden. Das Signal Reset (Zurücksetzen) setzt den Befehlszähler auf die Startadresse und das Befehlsschaltwerk auf den Zustand "Befehlsadresse aussenden". Das Signal Interrupt (Unterbrechung) bricht das Programm ab. Nach Ausführung des laufenden Befehls wird wie beim Unterprogrammaufruf der Befehlszählerstand gerettet und ein Unterprogramm gestartet. An dessen Ende steht der Rücksprungbefehl, der das unterbrochene Programm fortsetzt.



5. Konstanten

Das Modell sieht das Laden einer Konstanten in den Akkumulator vor. Das erste Wort des Befehls enthält den Funktionscode, im zweiten Wort befindet sich eine Konstante, die in den Akkumulator gebracht wird.



Damit sind alle 16 Möglichkeiten des 4-Bit-Funktionscodes ausgenutzt. Man unterscheidet 1-Wort-Befehle (Schieben und Rücksprung), 2-Wort-Befehle (Lade Konstante) und 3-Wort-Befehle (Laden, Speichern, Springen, logische und arithmetische Operationen).

1.9 Ein einfacher Modell-Mikrocomputer

Bild 40 zeigt einen Mikrocomputer in einer weiter vereinfachten Darstellung, der aus dem Rechenwerk Bild 37 und dem Steuerwerk Bild 39 zusammengesetzt wurde.
Er entspricht keinem handelsüblichen Gerät, sondern soll als Modell die allgemeinen Eigenschaften beschreiben.

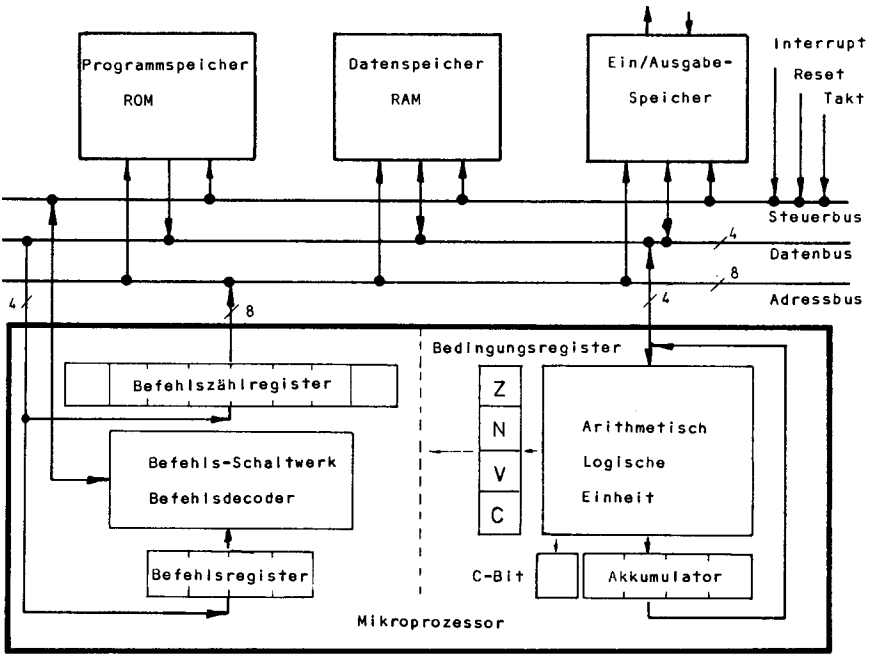


Bild 40: Modell-Mikrocomputer

Bild 41 zeigt die Befehlsliste. Für die Befehle wurde eine Kurzbezeichnung eingeführt, z.B. ADD für die Addition. Ein X in der Spalte Bedingungen zeigt an, daß der Befehl die Bits des Bedingungsregisters beeinflußt.

In der Spalte Zyklen steht die Zahl der für die Ausführung benötigten Takte, die eine Berechnung der Ausführungszeit des Programms erlauben. Bei den bedingten Sprüngen steht links der Wert für den Fall, daß nicht gesprungen wird, und rechts der Wert für den Fall des Sprunges. Bei einer angenommenen Zykluszeit von 1 µs (Takt 1 MHz) würde also die Ausführung des Additionsbefehls 4 Zyklen, also 4 µs dauern.

Neben den 16 Programmbefehlen gibt es zwei Signale, die den Programmablauf beeinflussen. Eine logische 0 auf der Leitung RESET veranlaßt, daß das Programm während der Ausführung des laufenden Befehls abgebrochen wird. Der Befehlszähler wird mit dem Inhalt der Speicherstellen 1111 1110 und 1111 1111 geladen. Dort steht die Anfangsadresse des nun zu startenden Programms. Eine logische 0 auf der Leitung INTERRUPT unterbricht das Programm

Befehl Symbol	binär	hexa	Bedingungen V	C	N	Z	Zahl der Zyklen	Wörter	Funktion
LAD	0000	0			X	X	4	3	Lade Akku
LDK	0001	1			X	X	2	2	Lade Konstante
SPE	0010	2					4	3	Speichere Akku
ADD	0011	3	X	X	X	X	4	3	Addiere Akku
SUB	0100	4	X	X	X	X	4	3	Subtrahiere Akku
UND	0101	5			X	X	4	3	logisches UND
ODR	0110	6			X	X	4	3	logisches ODER
EOD	0111	7			X	X	4	3	logisches XOR
SLR	1000	8					2	1	schiebe log. rechts
SCR	1001	9					2	1	schiebe zyc. rechts
SPR	1010	A					3	3	springe immer
SPP	1011	B					2/3	3	springe bei plus
SPM	1100	C					2/3	3	springe bei minus
SPN	1101	D					2/3	3	springe bei Null
SUP	1110	E					7	3	Unterprogrammsprung
RUP	1111	F					3	1	Rücksprung

Bild 41: Befehlsliste des Modell-Mikrocomputers

nach Beendigung des laufenden Befehls, rettet den Befehlszähler in die Speicherstellen 0111 1111 und 0111 1110 und lädt den Inhalt der Speicherstellen 1111 1100 und 1111 1101 als neue Befehlsadresse in den Befehlszähler.

Die meisten ausgeführten Mikroprozessoren kennen noch zwei Erweiterungen in der Adreßbehandlung, das Indexregister und das Stapel-Adreßregister (Stackpointer). Das Indexregister gestattet die Berechnung von Datenadressen durch Laden, Addieren und Subtrahieren. Das Stapel-Adreßregister enthält die laufende Adresse eines besonderen Speicherbereiches, der Adressen und Daten stapelt. Wie bei einem Spielkartenstapel kann man Daten oben auf den Stapel legen und von oben wieder entfernen. Auch für diesen Modell-Mikrocomputer wäre es sinnvoll, die Rücksprungadresse beim Unterprogrammsprung und beim Interrupt in solch einem Stapel abzulegen, da für den Fall eines Interrupts in einem Unterprogramm oder bei geschachtelten Unterprogrammaufrufen mehrere Rücksprungadressen gespeichert werden müssen.

Der acht bit breite Adreßbus erlaubt also die Adressierung von $2^8 = 256$ Speicherwörtern. Bild 42 zeigt den Adreßplan.

Adresse binär	hexa	Verwendung	Aufbau
0000 0000	00	Eingaberegister	Register
0000 0001	01	Ausgaberegister	Register
0000 0010 bis 0111 1101	02 bis 7D	Variablenspeicher	Lese-Schreib-Speicher (RAM)
0111 1110	7E	Rücksprungadresse bei Unterprogrammsprung und Interrupt	
0111 1111	7F		
1000 0000 bis 1111 1011	80 bis FB	Programmspeicher (Befehle und Konstanten)	Nur-Lese-Speicher (ROM)
1111 1100	FC	Speicher für die Sprungadresse bei Interrupt	
1111 1101	FD		
1111 1110	FE	Speicher für die Startadresse bei Reset	
1111 1111	FF		

Bild 42: Adreßplan des Modell-Mikrocomputers

Abschließend ein kleines Programmbeispiel:

Am Eingaberegister erscheinen laufend Bitkombinationen, die am Ausgaberegister wieder auszugeben sind. Ist das werthöchste Bit eine 1, so ist vorher der Wert zu komplementieren. Im Falle eines Interrupts ist mindestens 50 µs lang die Bitkombination 1111 auszugeben, bevor das Programm weiterläuft.
Zunächst wird das Hauptprogramm entwickelt.

1. Schritt: Beschreibung des Verfahrens mit Worten.
Am Anfang des Programms wird der Inhalt des Eingaberegisters in den Akkumulator gebracht. Das werthöchste Bit (Vorzeichen) beeinflußt das N-Bit des Bedingungsregisters. Ist es 1 (Vorzeichen minus), so wird der Akkumulatorinhalt durch ein XOR mit der Konstanten 1111 komplementiert. Ist es nicht gesetzt (Vorzeichen plus), so ist dieses Programmstück zu überspringen. Anschließend folgt das Speichern des Akkumulatorinhaltes in das Ausgaberegister und ein unbedingter Sprung zurück zum Anfang des Programms.

2. Schritt: Entwicklung eines Programmablaufplans
Bild 43 zeigt den Programmablaufplan mit den genormten Sinnbildern.

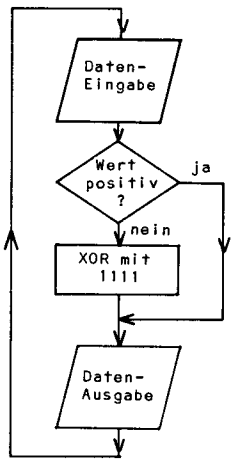


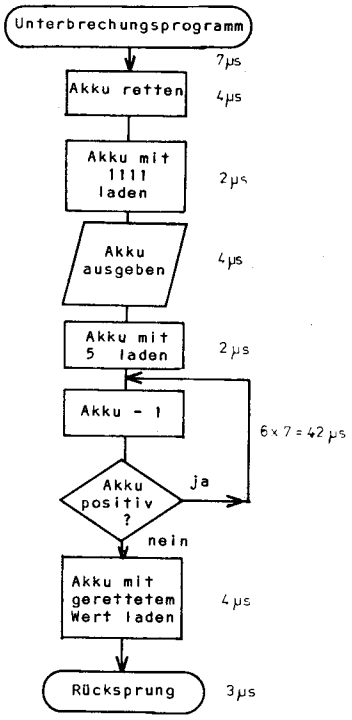
Bild 43: Programmablaufplan des Hauptprogramms

3. Schritt: Symbolisches Programm
Wie bereits gezeigt, besteht das Programm im Speicher aus binären Funktionscodes und binären Adressen. Da es jedoch in dieser Form nur schwer überschaubar ist, benutzt man bei der Programmierung für die Funktionscodes gerne die Symbole des Bildes 41 und für die Adressen frei wählbare Namen. Diese Art der Darstellung nennt man Assemblerschreibweise, da sie sich leicht von einem Übersetzungsprogramm - Assembler genannt - in die binäre Darstellung überführen läßt. Das Programm kann auf besonderen Formblättern nach Bild 44 entworfen werden.
Die Namen EIN, AUS, SUSI und MAX wurden frei erfunden. Unter der Speicherstelle KONST soll die Konstante $1111_2 = F_{16}$ gespeichert sein. Das Eingaberegister trägt die Bezeichnung EIN, das Ausgaberegister heißt AUS.

Name	Befehl	Operand	Bemerkung
1	7	13	19 25 30
S.U.S.I.	L.A.D.	E.I.N.	
	S.P.P.	M.A.X.	
	E.O.D.	K.O.N.S.T.	
M.A.X.	S.P.E.	A.U.S.	
	S.P.R.	S.U.S.I.	

Bild 44: Symbolisches Hauptprogramm

Der 4.Schritt, die Umwandlung des symbolischen Programms in den Binärcode, wird zusammen mit dem nun zu erstellenden Unterbrechungsprogramm vorgenommen.
Für das Unterbrechungsprogramm spielen Zeitbetrachtungen eine wesentliche Rolle. Durch das Interrupt-Signal wird der Befehlszähler gerettet und mit der Anfangsadresse des Unterbrechungsprogramms geladen (7 µs). Der Inhalt des Akkumulators wird im neuen Programm zerstört und muß daher zunächst in den Variablenspeicher gerettet werden (4µs). Dann ist die Konstante 1111 in den Akkumulator zu bringen (2 µs) und anschließend auszugeben (4 µs). Dieser Zustand des Ausgaberegisters soll nun 50 µs lang erhalten bleiben. Dazu wird der Akkumulator als Rückwärtszähler



Name	Befehl	Operand	Bemerkung
1	7	13	19 25 30
O.T.T.O.	S.P.E.	R.E.T.T.	
	L.D.K.	1.1.1.1.	
	S.P.E.	A.U.S.	
	L.D.K.	0.1.0.1.	
P.A.U.L.	S.U.B.	E.I.N.S.	
	S.P.P.	P.A.U.L.	
	L.A.D.	R.E.T.T.	
	R.U.P.		

Bild 46:
Symbolisches Unterbrechungsprogramm

Bild 45: Programmablaufplan des Unterbrechungsprogramms

benutzt, der von einem Ausgangswert so lange um 1 vermindert wird, bis er negativ ist. Die beiden Befehle "Subtrahiere" und "Springe bei positiv" benötigen zusammen 7 µs. Bei sechsmaliger Ausführung ergibt sich eine Zeitverzögerung von 42 µs. Das vorherige Laden des Akkumulators mit der Konstanten 5 (2 µs), das abschließende Rückladen des alten Akkumulatorinhaltes (4 µs) und der Rücksprung (3 µs) bringen weitere 9 µs Zeitverzögerung.

Bild 45 zeigt den Programmablaufplan mit den Ausführungszeiten der Befehle und Bild 46 das symbolische Programm.

5. Schritt: Speicherbelegungsplan
Bild 47 zeigt links die binäre und rechts die verkürzte hexadezimale Darstellung des von den Programmen benötigten Speichers.

Symbol	binär Adresse	binär Inhalt	hexadezimal Adresse	hexadezimal Inhalt	Bemerkung
EIN	0000 0000	variabel	00	variabel	Eingaberegister
AUS	0000 0001	variabel	01	variabel	Ausgaberegister
RETT	0000 0010	variabel	02	variabel	Rettung des Akkus
	0111 1110	variabel	7E	variabel	Rücksprungadresse
	0111 1111	variabel	7F	variabel	Rücksprungadresse
KONST	1000 0000	1111	80	F	Konstante
EINS	1000 0001	0001	81	1	Konstante
SUSI	1000 0010	0000	82	0	LAD Startadresse
	1000 0011	0000	83	0	EIN Hauptprogramm
	1000 0100	0000	84	0	
	1000 0101	1011	85	B	SPP
	1000 0110	1000	86	8	MAX
	1000 0111	1001	87	8	
	1000 1000	0111	88	7	EOD
	1000 1001	1000	89	8	KONST
	1000 1010	0000	8A	0	
MAX	1000 1011	0010	8B	2	SPE
	1000 1100	0000	8C	0	AUS
	1000 1101	0001	8D	1	
	1000 1110	1010	8E	A	SPR
	1000 1111	1000	8F	8	SUSI
	1001 0000	0010	90	2	
OTTO	1001 0001	0010	91	2	SPE Unterbrechungs-
	1001 0010	0000	92	0	RETT programm
	1001 0011	0010	93	2	
	1001 0100	0001	94	1	LDK
	1001 0101	1111	95	F	1111
	1001 0110	0010	96	2	SPE
	1001 0111	0000	97	0	AUS
	1001 1000	0001	98	1	
	1001 1001	0001	99	1	LDK
	1001 1010	0101	9A	5	0101
PAUL	1001 1011	0100	9B	4	SUB
	1001 1100	1000	9C	8	EINS
	1001 1101	0001	9D	1	
	1001 1110	1011	9E	B	SPP
	1001 1111	1001	9F	9	PAUL
	1010 0000	1011	A0	B	
	1010 0001	0000	A1	0	LAD
	1010 0010	0000	A2	0	RETT
	1010 0011	0010	A3	2	
	1010 0100	1111	A4	F	RUP
	1111 1100	1001	FC	9	Interruptadresse
	1111 1101	0001	FD	1	OTTO
	1111 1110	1000	FE	8	Startadresse
	1111 1111	0010	FF	2	SUSI

Bild 47: Speicherbelegungsplan der Beispielprogramme

Übungen zum Abschnitt 1.7 bis 1.9

1.Aufgabe
Es ist ein Programm mit Programmablaufplan zu entwerfen, das von Null ausgehend den Akkumulator laufend um 1 erhöht und ausgibt.

2.Aufgabe
Eine 4-Bit-Mikrocomputereinheit habe eine Befehlsliste entsprechend Bild 41.
Eine Speicherstelle OTTO habe den hexadezimalen Wert F, eine Speicherstelle SUSI den Wert 8.
Welchen Inhalt nimmt die Speicherstelle PAUL nach Ablauf des folgenden Programms an?

```
LDK   1
UND   OTTO
ADD   SUSI
SPE   PAUL
```

3.Aufgabe
Für einen 4-Bit-Mikrocomputer mit einer Befehlsliste nach Bild 41 entwerfe man

einen Programmablaufplan,
ein symbolisches Programm und
ein hexadezimales Programm für folgendes Problem:

In dem Ausgaberegister mit dem symbolischen Namen AUS und der Adresse 01 sind nacheinander die Zahlen 0, 2, 4 und 6 auszugeben. Dann soll das Programm in einer Warteschleife auf einen Interrupt warten. Man beachte, daß die Addition von 6 + 2 = 8 in der vorzeichenbehafteten Arithmetik ein negatives Ergebnis liefert!

2. Der Aufbau ausgeführter Mikrocomputer

Der einfache Modell-Mikrocomputer des vorigen Abschnitts bestand aus:
- dem Mikroprozessor mit Rechenwerk und Steuerwerk,
- einem Programmspeicher,
- einem Datenspeicher und
- einem Speicherbaustein für die Ein/Ausgabe von Daten.

Nicht erwähnt wurden zusätzliche Schaltungen wie Taktgeber, Adreßdecoder und Treiber.

Das Angebot an Bauelementen für Mikrocomputer ist groß. Je nach Länge des Datenwortes unterscheidet man 4-Bit, 8-Bit, 12-Bit und 16-Bit-Mikroprozessoren mit den entsprechenden Speicher- und Ein/Ausgabe-Bausteinen. Am häufigsten werden zur Zeit 8-Bit-Systeme verwendet. Sie verarbeiten die Daten acht bit parallel. Der Adreßbus enthält 16 Adreßleitungen. Dies bedeutet, daß 2^{16} = 65 536 Bytes adressiert werden können. Mit der Einheit 1 K = 1024 sind es 64 K Bytes.

Bei Ein/Ausgabe-Bausteinen unterscheidet man zwischen Parallel- und Serienschnittstellen. Sie liegen auf der einen Seite am parallel arbeitenden Datenbus und stellen auf der anderen Seite die Verbindung zu parallel oder seriell arbeitenden Geräten her. Zur Anpassung an die verschiedenen Aufgaben sind ihre Funktionen programmierbar.

Beim Bau eines Mikrocomputers müssen sowohl die Schaltungen (Hardware) als auch die Programme (Software) entwickelt und getestet werden. Dazu bedient man sich sogenannter Entwicklungssysteme. Dies sind besondere Mikrocomputer mit fest eingebauten Programmen, dem Betriebssystem. Der Entwickler kann mit ihnen:

- über eine Tastatur Programme hexadezimal oder symbolisch in einen Schreib-Lese-Speicher (RAM) eingeben,
- den Lauf der Programme schrittweise verfolgen,
- die Programme mit den zu steuernden Geräten laufen lassen,
- die Programme ändern und auf einem Bildschirm oder Drucker ausgeben und
- die fertigen Programme aus dem RAM in ein PROM (Programmierbaren Nur-Lese-Speicher) bringen.

Für die Auswahl der drei folgenden Beispiele ausgeführter Mikrocomputer waren folgende Gesichtspunkte maßgebend:

1. Sie sollten häufig in der Industrie (Motorola 6800 und Intel 8080 bzw. 8085) oder in Hobbygeräten (KIM-1) verwendet werden.
2. Es sollten einfache und billige Bausätze (Kits) oder fertige Entwicklungssysteme mit mindestens einer hexadezimalen Programmeingabe zur Verfügung stehen.

3. Als Ergänzung sollte genügend - auch deutschsprachige - Literatur mit Beispielen, Programmen und technischen Beschreibungen vorhanden sein.

Aus Platz- und Zeitgründen konnte eine große Anzahl ebenso guter, wichtiger und preisgünstiger Systeme nicht aufgenommen werden.

Für den Test der Beispielprogramme kann eine Schaltung nach Bild 48 verwendet werden. Sie gestattet über eine Parallelschnittstelle die Anzeige von acht Bits über Leuchtdioden (LEDs) und die Eingabe von acht Bits über Kippschalter.

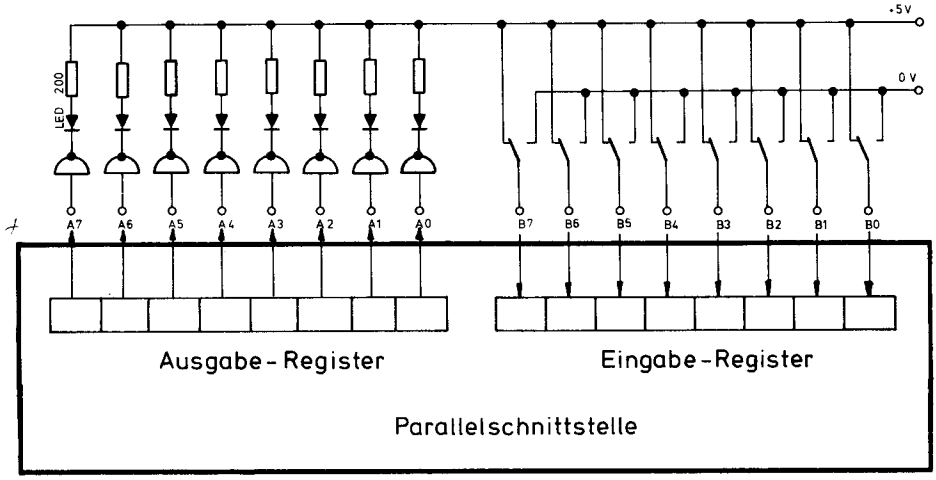


Bild 48: Schaltung zur Ein/Ausgabe von Daten

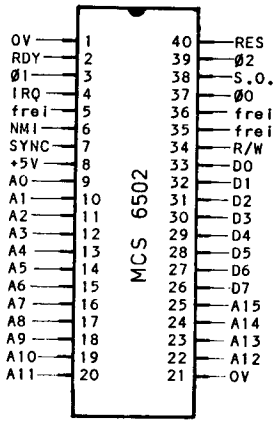
2.1 Der Mikrocomputer KIM-1

Mit diesem Übungsgerät kann man Programme in einen Benutzerbereich eingeben und anschließend ausführen lassen.
Bild 49 zeigt die Anschlußbelegung und die Organisation des Mikroprozessors MCS 6502, der im Mikrocomputer KIM-1 enthalten ist.

Der Mikroprozessor-Baustein ist 5 cm lang und 1,5 cm breit. Die Siliziumscheibe mit der eigentlichen Schaltung dagegen nur etwa 5 x 5 mm.

Bild 50 zeigt einen Übersichtsplan des Gerätes. Dabei sind nur die dem Programmierer zugänglichen Register des Mikroprozessors eingezeichnet.

Zwei Mehrzweck-Bausteine enthalten in einem ROM-Bereich das Betriebssystem, das Befehle und Daten des Benutzers von einem Tastenfeld in einen RAM-Bereich übernimmt und gleichzeitig an einem Anzeigefeld anzeigt. Die Eingabe und Anzeige erfolgen hexadezimal. Zusätzlich lassen sich ein Kassettenrecorder und ein Fernschreiber anschließen. Dem Benutzer stehen 1 K RAM und 15 Ein/Ausgabeleitungen zur freien Verfügung.



A0 bis A15	Adress-Bus
D0 bis D7	Daten-Bus
RDY	Fertig Programmverzögerung
Ø0 Ø1 Ø2	Takte
IRQ	maskierbarer Interrupt
NMI	nicht maskierbarer Interrupt
SYNC	Operationscode holen
RES	Rücksetzen und Start
S.O.	Setze Überlauf
R/W	Lesen/Schreiben

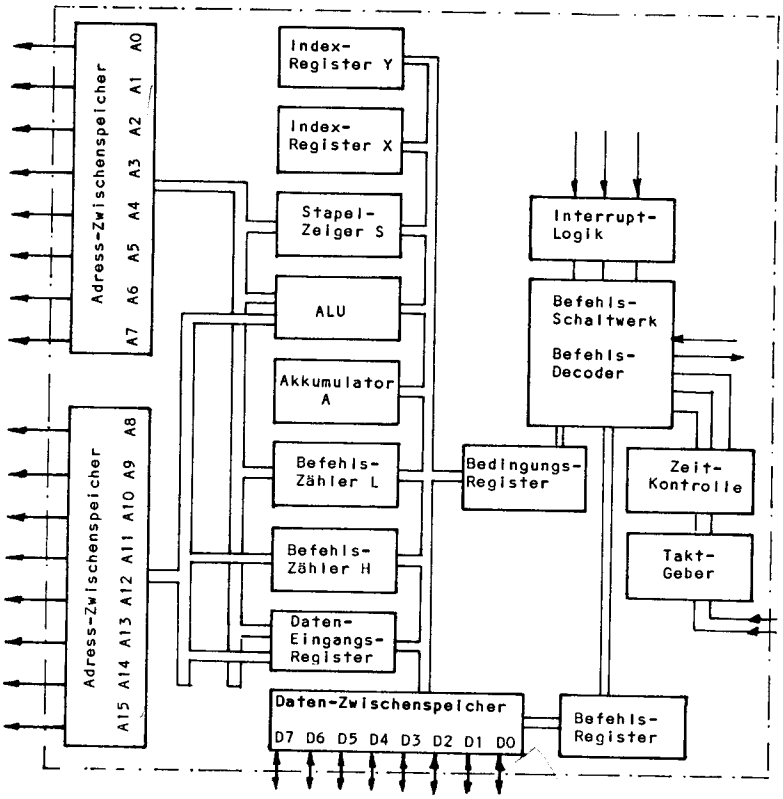


Bild 49: Mikroprozessor MCS 6502
(Nach Unterlagen der Firma MOS Technology)

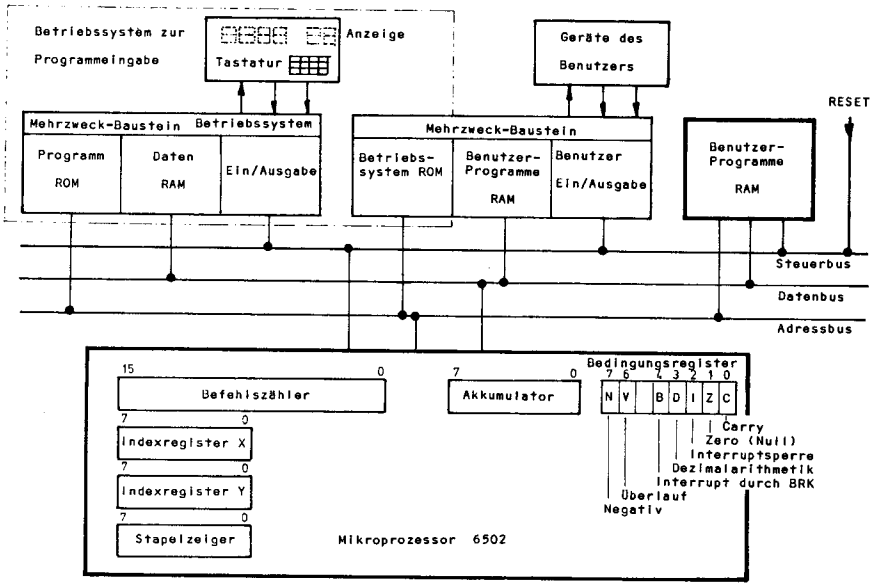


Bild 50: Übersichtsplan des KIM-1

Der Mikroprozessor enthält folgende dem Programmierer zugängliche Register:

einen 16-Bit-Befehlszähler,
zwei 8-Bit-Indexregister (X und Y) für Adreßrechnungen,
einen 8-Bit-Stapelzeiger zur Adressierung des Datenstapels,
einen 8-Bit-Akkumulator und
ein 8-Bit-Bedingungsregister.

Anlage 5 enthält die Befehlsliste mit allen Adressierungsmöglichkeiten.
Bild 51 zeigt den Adreßplan des Benutzerbereiches.

Adresse	Gerät	Verwendung
17EB ⋮ 1780	RAM	frei für Benutzer-Programme
1703 1702 1701 1700	E/A	Richtungsregister B-Seite Datenregister B-Seite Richtungsregister A-Seite Datenregister A-Seite
03FF ⋮ 0200	RAM	frei für Benutzer-Programme
01FF ⋮ 0100	RAM	Stapel auch für Benutzer
00EE ⋮ 0000	RAM	frei für Benutzer-Programme Seite 0

Bild 51: Adreßplan des Benutzerbereiches

Das folgende einführende Beispiel testet die Ein/Ausgabeschaltung des Bildes 48, die als Gerät des Benutzers angeschlossen wurde. Dabei werden beliebige Daten von den Kippschaltern gelesen und mit den Leuchtdioden angezeigt.
Bild 52 zeigt das Programm rechts in Assemblerschreibweise und links in hexadezimaler Darstellung.

```
LISTEN-ASSEMBLER KIM-1 VERSION 1
  1.                      AUS    =$1700        DATENREGISTER A-SEITE
  2.                      RAU    =$1701        RICHTUNGSREGISTER A-SEITE
  3.                      EIN    =$1702        DATENREGISTER B-SEITE
  4.                      REI    =$1703        RICHTUNGSREGISTER B-SEITE
  5.                             *=$0000
  6.      0000  A9        START  LDA    #$00    PROGRAMMIERUNG DER E/A
                00
  7.      0002  8D               STA    REI     B-SEITE IST EINGANG
                03
                17
  8.      0005  A9               LDA    #$FF
                FF
  9.      0007  8D               STA    RAU     A-SEITE IST AUSGANG
                01
                17
 10.      000A  AD        SUSI   LDA    EIN     LADE AKKUMULATOR
                02
                17
 11.      000D  8D               STA    AUS     SPEICHERE AKKUMULATOR
                00
                17
 12.      0010  4C               JMP    SUSI    SPRINGE ZUM BEFEHL SUSI
                0A
                00
 13.                            .END            ENDE DES ASSEMBLERPROGRAMMS

NAMENSLISTE
AUS    1700
RAU    1701
EIN    1702
REI    1703
START  0000
SUSI   000A
```

Bild 52: Einführendes Beispiel KIM-1

Das Bild zeigt die Übersetzungsliste eines Cross-Assemblers, der auf einem Großrechner vom Typ UNIVAC 1108 die symbolische Eingabe (rechts) in hexadezimalen Code (links) übersetzt hat. Die Eingabezeilen sind am linken Rand durchnumeriert. Die Zeilen 1 bis 4 legen die Adressen der Ein/Ausgaberegister fest. Die Zeile 5 bestimmt, daß das Programm mit der Adresse 0000 beginnen soll. Mit den Zeilen 6 und 7 wird die Konstante 00_{16} = 0000 0000 binär in den Akkumulator geladen und anschließend in das Datenrichtungsregister der B-Seite gebracht. Damit sind diese Leitungen Eingänge. Mit den Befehlen der Zeilen 8 und 9 werden die Leitungen der A-Seite zu Ausgängen erklärt.Die Befehle der Zeilen 10, 11 und 12 bilden eine Schleife, in der fortlaufend der Inhalt der Eingabe in den Akkumulator gebracht und anschließend sofort wieder ausgegeben wird. Das END der Zeile 13 schließt das Programm ab.

Die vierstelligen Hexadezimalzahlen der 2. Spalte geben die Adressen der Befehle an (Adreß-Zähler); die Anfangsadresse wurde mit der 5.Zeile auf 0000 gesetzt. Die zweistelligen Hexadezimalzahlen rechts daneben bilden das Programm. Sie enthalten Funktionscodes, Konstanten und Adressen. Dieses Programm ist nun mit Hilfe des Betriebssystems in den Mikrocomputer einzugeben und zu starten. Da es in einer unendlichen Schleife läuft, kann es nur durch RESET oder einen Interrupt abgebrochen werden. Man beachte, daß nur sieben Eingabeleitungen zur Verfügung stehen. Die Leitung B6 ist nicht angeschlossen.

2.2 Ein Mikrocomputer mit dem Prozessor Motorola 6800

Als Beispiel dient der Entwicklungs-Bausatz MEK 6800 D2. Mit diesem Übungs- und Entwicklungsgerät kann man Programme in einen Benutzerbereich eingeben und anschließend ausführen lassen.

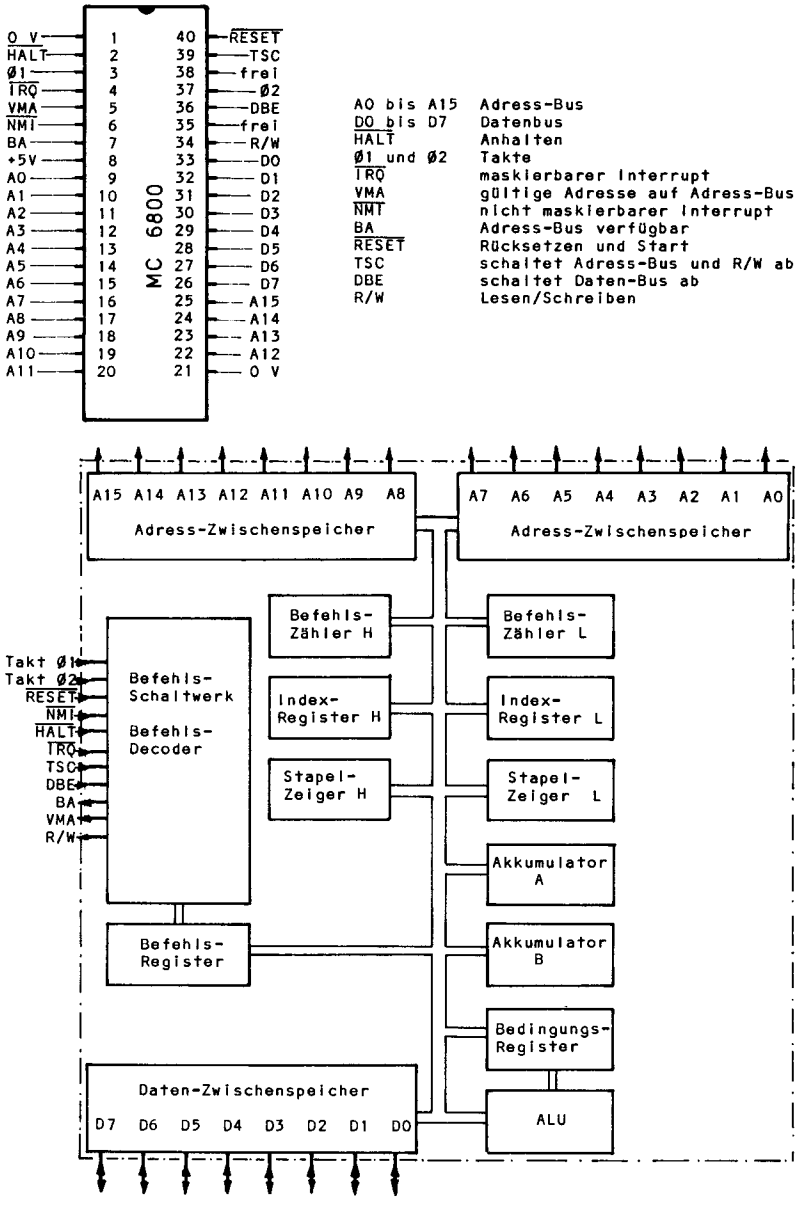


Bild 53: Mikroprozessor MC 6800
(Nach Unterlagen der Firma Motorola)

Bild 53 zeigt die Anschlußbelegung und die Organisation des Mikroprozessors Motorola MC 6800.

Der Baustein ist 5 cm lang und 1,5 cm breit; die Siliziumscheibe mit der eigentlichen Schaltung dagegen nur etwa 5 x 5 mm. Bild 54 zeigt einen Übersichtsplan des Gerätes. Dabei sind in den Mikroprozessor nur die dem Programmierer zugänglichen Register eingezeichnet.

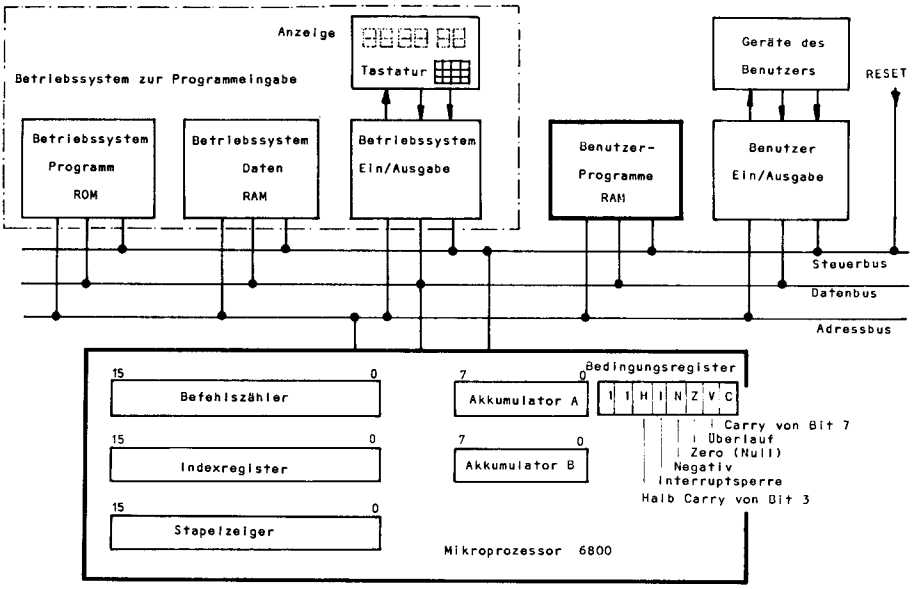


Bild 54: Übersichtsplan des MEK 6800 D2

Ein ROM enthält das Betriebssystem, das zusätzlich noch einen RAM-Speicher und zwei Ein/Ausgabe-Bausteine benötigt. Der Benutzer gibt seine Kommandos und sein Programm über eine Tastatur ein; die Werte werden hexadezimal angezeigt. Die Programme und Daten lassen sich auf einem Kassettenrecorder speichern. Dem Benutzer stehen 1/2 K RAM und 20 Ein/Ausgabeleitungen zur Verfügung.

Der Mikroprozessor enthält folgende dem Programmierer zugängliche Register:

einen 16-Bit-Befehlszähler,
ein 16-Bit-Indexregister für Adreßrechnungen,
einen 16-Bit-Stapelzeiger zur Adressierung des Datenstapels,
zwei 8-Bit-Akkumulatoren (A und B) und
ein 8-Bit-Bedingungsregister.

Anlage 6 enthält die Befehlsliste mit allen Adressierungsmöglichkeiten.
Bild 55 zeigt den Adreßplan des Benutzerbereiches.
Der für den Benutzer vorgesehene RAM-Bereich kann mit zusätzlichen Speicherschaltungen erweitert werden, die an den bestehenden Bausatz anzuschließen sind. Von dieser Möglichkeit wurde jedoch bei dem System kein Gebrauch gemacht.

Adresse	Gerät	Verwendung
8007 8006 8005 8004	E/A	Kontrollregister B-Seite Daten-/Richtungs-Register B-Seite Kontrollregister A-Seite Daten-/Richtungs-Register A-Seite
01FF ⋮ 0000	RAM	frei für Benutzer-Programme

Bild 55: Adreßplan des Benutzerbereiches

Das folgende einführende Beispiel testet die Ein/Ausgabeschaltung des Bildes 48, die als Gerät des Benutzers angeschlossen wurde. Dabei werden beliebige Daten von den Kippschaltern gelesen und mit den Leuchtdioden angezeigt.

Bild 56 zeigt das Programm rechts in Assemblerschreibweise und links in hexadezimaler Darstellung.

```
LISTEN-ASSEMBLER M6800 VERSION 1
   1.                        AUS    EQU   $8004   DATEN/RICHTUNG   A-SEITE
   2.                        KONA   EQU   $8005   KONTROLLREGISTER A-SEITE
   3.                        EIN    EQU   $8006   DATEN/RICHTUNG   B-SEITE
   4.                        KONB   EQU   $8007   KONTROLLREGISTER B-SEITE
   5.                               ORG   $0000
   6.        0000  7F      START  CLR   KONA     PROGRAMMIERUNG A-SEITE
                 80
                 05
   7.        0003  86             LDA A #$FF
                 FF
   8.        0005  B7             STA A AUS      A-SEITE IST AUSGANG
                 80
                 04
   9.        0008  C6             LDA B #$04
                 04
  10.        000A  F7             STA B KONA     AUS AUF DATEN GESTELLT
                 80
                 05
  11.        000D  7F             CLR   KONB     PROGRAMMIERUNG B-SEITE
                 80
                 07
  12.        0010  7F             CLR   EIN      B-SEITE IST EINGANG
                 80
                 06
  13.        0013  F7             STA B KONB     EIN AUF DATEN GESTELLT
                 80
                 07
  14.        0016  B6      SUSI   LDA A EIN      LADE AKKUMULATOR A
                 80
                 06
  15.        0019  B7             STA A AUS      SPEICHERE AKKUMULATOR A
                 80
                 04
  16.        001C  7E             JMP   SUSI     SPRINGE ZUM BEFEHL SUSI
                 00
                 16
  17.                             END            ENDE DES ASSEMBLER-PROGRAMMS

NAMENSLISTE
AUS   8004
KONA  8005
EIN   8006
KONB  8007
STAR  0000
SUSI  0016
```

Bild 56: Einführendes Beispiel 6800

Das Bild zeigt die Übersetzungsliste eines Cross-Assemblers, der auf einer Großrechenanlage vom Typ UNIVAC 1108 die symbolische Eingabe (rechts) in hexadezimalen Code (links) übersetzt hat.

Die Eingabezeilen sind am linken Rand durchnumeriert. Die Zeilen 1 bis 4 legen die Adressen der Ein/Ausgaberegister fest. Die Zeile 5 bestimmt, daß das Programm mit der Adresse 0000 beginnen soll.

Die Zeilen 6 bis 9 legen die A-Seite des Ein/Ausgabe-Bausteins als Ausgabe fest, die Zeilen 10 bis 13 die B-Seite als Eingang. Die Befehle der Zeilen 14, 15 und 16 bilden eine Schleife, in der fortlaufend der Inhalt der Eingabe in den Akkumulator gebracht und anschließend sofort wieder ausgegeben wird. Das END der Zeile 17 schließt das Programm ab.

Die vierstelligen Hexadezimalzahlen der 2.Spalte geben die Adressen der Befehle an (Adreß-Zähler); die Anfangsadresse wurde mit dem ORG-Befehl auf 0000 gesetzt. Die zweistelligen Hexadezimalzahlen rechts daneben in der 3.Spalte bilden das Programm. Sie enthalten Funktionscodes, Konstanten und Adressen.

Dieses Programm wird nun mit Hilfe des Betriebssystems in den Benutzer-RAM gebracht und anschließend gestartet. Da es eine unendliche Schleife enthält, kann es nur durch RESET oder einen Interrupt abgebrochen werden.

2.3 Mikrocomputer mit den Prozessoren INTEL 8080 und 8085

Als Beispiele dienen die Entwicklungs-Bausätze SDK 8080 und SDK 8085. Mit diesen Geräten kann man Programme in einen Benutzer-Bereich eingeben und anschließend ausführen lassen. Bild 57 zeigt die Anschlußbelegung und die Organisation des Mikroprozessors 8080; der 8085 ist ähnlich aufgebaut.

Der Baustein ist 5 cm lang und 1,5 cm breit; die Siliziumscheibe mit der eigentlichen Schaltung dagegen nur etwa 5 x 5 mm. Bild 58 zeigt den Übersichtsplan des SDK 8080, der SDK 8085 ist ähnlich aufgebaut. In den Mikroprozessor wurden nur die dem Programmierer zugänglichen Register eingezeichnet.

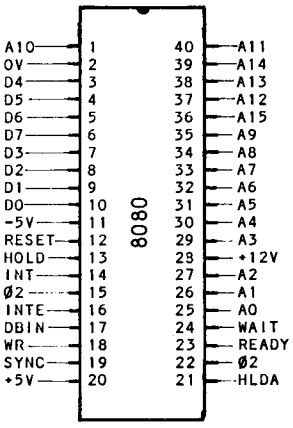
Der Mikroprozessor 8080/8085 enthält folgende dem Programmierer zugängliche Register:

einen 16-Bit-Befehlszähler,
einen 16-Bit-Stapelzeiger zur Adressierung des Datenstapels,
drei 16-Bit-Indexregister für Datenadressen, wahlweise auch einzeln als 8-Bit-Register verwendbar,
einen 8-Bit-Akkumulator und
ein 8-Bit-Bedingungsregister.

Die Befehlslisten der einzelnen Abschnitte und der Anlage 7 enthalten Ausführungszeiten für die Befehle. Da es für die Prozessoren 8080 und 8085 verschiedene Ausführungen gibt, sind für die praktische Arbeit die Datenblätter der Hersteller heranzuziehen.

Bild 59a zeigt für den SDK 8080 den Adreßplan des Benutzerbereiches, Bild 59b den Adreßplan des SDK 8085.

Der Entwicklungs-Bausatz SDK 8080 besteht aus einem ROM mit den Programmen des Betriebssystems, einem RAM-Bereich und einer Parallelschnittstelle für den Benutzer, der seine Befehle über



A0 bis A15	Adress-Bus
D0 bis D7	Daten-Bus
RESET	Rücksetzen und Start
HOLD	Anhalten
INT	Interrupt
INTE	Interrupt - Freigabe
DBIN	Daten-Bus Eingabe
WR	Schreiben
SYNC	Operationscode holen
WAIT	Prozessor wartet
READY	Bereit
HLDA	Prozessor hält an

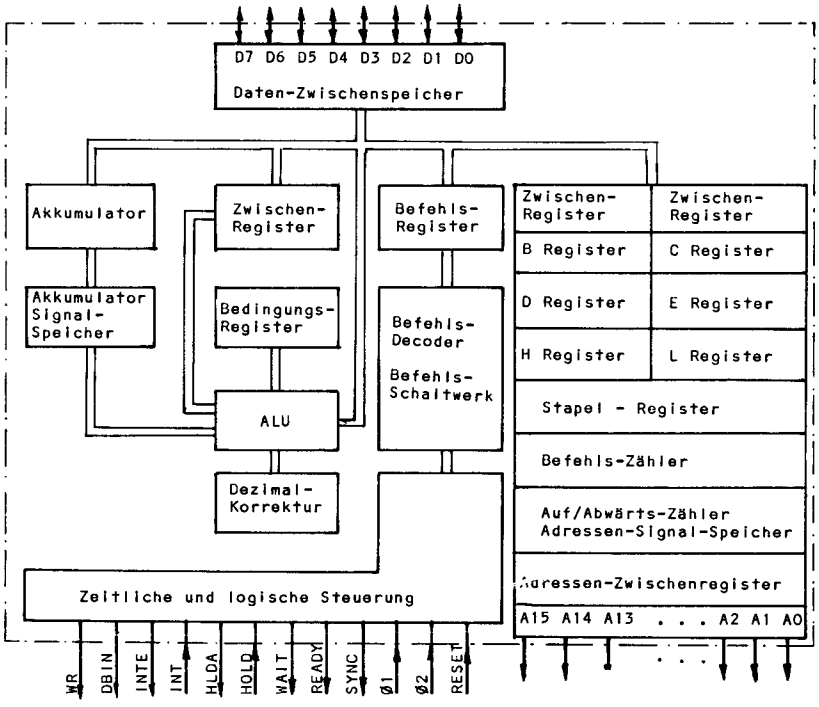


Bild 57: Mikroprozessor INTEL 8080
(Nach Unterlagen der Firma Intel)

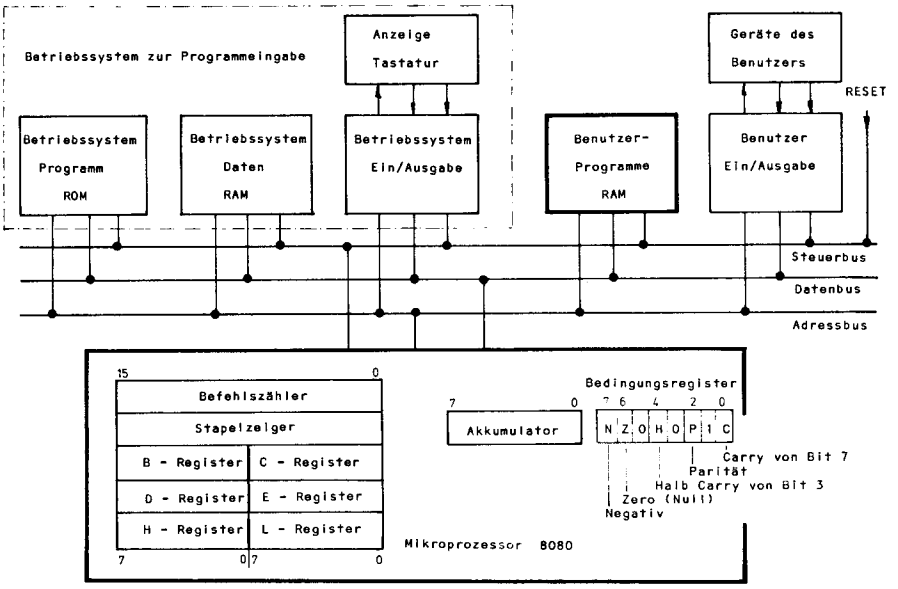


Bild 58: Übersichtsplan SDK 8080 (SDK 8085)

Adresse	Gerät	Verwendung
13FF ⋮ 1000	RAM	frei für Benutzer-Programme je nach Ausbau
		Parallelschnittstelle 8255
14	E/A	Kanal A
15	E/A	Kanal B
16	E/A	Kanal C
17	E/A	Steuer-Register

Bild 59a: Adreßplan des SDK 8080

Adresse	Gerät	Verwendung
20C7 ⋮ 2000	RAM	frei für Benutzer-Programme
28FF ⋮ 2800	RAM	frei für Benutzer-Programme je nach Ausbau
		Parallelschnittstelle 8355
00	E/A	Kanal A Datenregister
01	E/A	Kanal B Datenregister
02	E/A	Kanal A Richtungsregister
03	E/A	Kanal B Richtungsregister
		Parallelschnittstelle 8155
20	E/A	Steuer-Register
21	E/A	Kanal A
22	E/A	Kanal B
23	E/A	Kanal C

Bild 59b: Adreßplan des SDK 8085

einen Fernschreiber (Teletype) hexadezimal eingibt. Im Endausbau stehen 1 K RAM und 24 Ein/Ausgabeleitungen zur Verfügung.

Bei dem Entwicklungs-Bausatz SDK 8085 kann der Benutzer seine Programme auch über eine eingebaute Hexadezimaltastatur eingeben. Es stehen zwei verschiedene Parallelschnittstellen zur Verfügung.

Die folgenden einführenden Beispiele Bild 60a bis 60c testen die Bausätze mit Hilfe einer Ein/Ausgabeschaltung nach Bild 48, die

```
LISTEN-ASSEMBLER INTEL 8080 VERSION 1

  1.                          ORG   1300H
  2.        1300  3E    START MVI   A,8BH    LADE STEUERWORT
                  8B
  3.        1302  D3          OUT   17H      PROGRAMMIERUNG DER E/A
                  17
  4.        1304  DB    SUSI  IN    15H      LADE AKKUMULATOR
                  15
  5.        1306  D3          OUT   14H      SPEICHERE AKKUMULATOR
                  14
  6.        1308  C3          JMP   SUSI     SPRUNG ZUM BEFEHL SUSI
                  04
                  13
  7.                          END            ENDE DES ASSEMBLERPROGRAMMS

NAMENSLISTE
START 1300
SUSI  1304
```

Bild 60a: Einführendes Beispiel SDK 8080

```
LISTEN-ASSEMBLER INTEL 8080 VERSION 1

  1.                          ORG   2000H
  2.        2000  3E    START MVI   A,01H    LADE STEUERWORT
                  01
  3.        2002  D3          OUT   20H      PROGRAMMIERUNG DER E/A
                  20
  4.        2004  DB    SUSI  IN    22H      LADE AKKUMULATOR
                  22
  5.        2006  D3          OUT   21H      SPEICHERE AKKUMULATOR
                  21
  6.        2008  C3          JMP   SUSI     SPRUNG ZUM BEFEHL SUSI
                  04
                  20
  7.                          END            ENDE DES ASSEMBLERPROGRAMMS

NAMENSLISTE
START 2000
SUSI  2004
```

Bild 60b: Einführendes Beispiel SDK 8085 Parallelschnittstelle 8155

```
LISTEN-ASSEMBLER INTEL 8080 VERSION 1

  1.                          ORG   2000H
  2.        2000  3E    START MVI   A,0FFH   LADE STEUERWORT
                  FF
  3.        2002  D3          OUT   02H      RICHTUNGSREGISTER KANAL A
                  02
  4.        2004  3E          MVI   A,00H    LADE STEUERWORT
                  00
  5.        2006  D3          OUT   03H      RICHTUNGSREGISTER KANAL B
                  03
  6.        2008  DB    SUSI  IN    01H      LADE AKKUMULATOR
                  01
  7.        200A  D3          OUT   00H      SPEICHERE AKKUMULATOR
                  00
  8.        200C  C3          JMP   SUSI     SPRUNG ZUM BEFEHL SUSI
                  08
                  20
  9.                          END            ENDE DES ASSEMBLERPROGRAMMS

NAMENSLISTE
START 2000
SUSI  2008
```

Bild 60c: Einführendes Beispiel SDK 8085 Parallelschnittstelle 8355

als Gerät des Benutzers angeschlossen wurde. Dabei werden beliebige Daten von den Kippschaltern gelesen und mit Leuchtdioden angezeigt.

Da die Befehlssätze der Mikroprozessoren 8080 und 8085 bis auf zwei Befehle übereinstimmen, werden alle folgenden Beispiele nur noch für den Bausatz SDK 8085 aufgestellt. Für den SDK 8080 sind nur die Adressen zu ändern, die Befehle bleiben.

Bild 60a zeigt die Übersetzungsliste eines Cross-Assemblers, der auf einer Großrechenanlage vom Typ UNIVAC 1108 die symbolische Eingabe (rechts) in hexadezimalen Code (links) übersetzt hat. Die Eingabezeilen sind am linken Rand durchnumeriert. Die 1.Zeile legt fest, daß das Programm mit der Adresse 1300 beginnen soll. Die 2.Zeile lädt den Akkumulator mit einem Steuerwort, das durch den nächsten Befehl (3.Zeile) in das Steuerregister der Ein/Ausgabe gebracht wird.

Die Befehle 4,5 und 6 bilden eine Schleife, in der fortlaufend der Inhalt der Eingabe in den Akkumulator gebracht und anschließend sofort wieder ausgegeben wird. Das END der Zeile 7 schließt das Programm ab.

Die vierstelligen Hexadezimalzahlen der 2.Spalte geben die Adressen der Befehle an (Adreßzähler); die Anfangsadresse wurde mit dem ORG-Befehl auf 1300 gesetzt. Die zweistelligen Hexadezimalzahlen daneben in Spalte 3 bilden das Programm. Sie enthalten Funktionscodes, Konstanten und Adressen sowie Nummern der Ein/Ausgabekanäle. Dieses Programm ist nun mit Hilfe des Betriebssystems einzugeben und zu starten. Da es in einer unendlichen Schleife läuft, kann es nur durch ein RESET oder Interrupt abgebrochen werden.

2.4 Die Programmierung von Mikrocomputern A

Am Anfang der Programmierung steht das Problem; das Ziel ist es, Befehle und Konstanten in Form von binären Nullen und Einsen in den Programmspeicher zu bringen.

Auch die Konstruktion eines technischen Gerätes beginnt damit, die Anforderungen, die an das Gerät zu stellen sind, genau festzulegen. In der Programmierung ist dies die Problemanalyse. Soll das Gerät mit anderen in einem System zusammenarbeiten, so sind die Beziehungen der Geräte untereinander zu klären; in der Datenverarbeitung ist dies die Systemanalyse.

Die eigentliche Entwurfsarbeit beginnt bei der Geräteentwicklung mit dem Blockschaltplan, der den Zusammenhang der Baugruppen (Netzteil, Verstärker, Eingangsstufe usw.) zeigt, die durch genormte Symbole dargestellt werden. Wichtig ist hierbei, daß die Struktur (Aufbau, innere Gliederung, Zusammenhang) des Gerätes und seiner Baugruppen leicht zu überschauen ist. Auf die Programmierung übertragen bedeutet dies, daß ein Übersichtsplan zu entwerfen ist, der sofort den Zusammenhang der Programmteile und die Funktion des Gesamtprogramms erkennen läßt. Die herkömmlichen Programmablaufpläne versagen hier, da ihre Symbole nur die Wirkung eines einzelnen Befehls beschreiben. Das in der Datenverarbeitung in zunehmenden Maße eingeführte Programmierverfah-

ren "Strukturierte Programmierung" beschreibt zunächst den Aufbau und die Wirkungsweise eines Programms in einem Struktogramm, hier in Anlehnung an die Elektrotechnik Programmblockplan genannt.
Bild 61 zeigt den Programmblockplan des einführenden Beispiels.

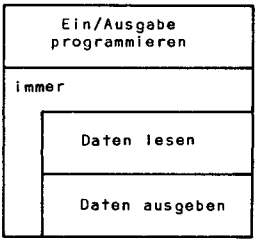


Bild 61: Programmblockplan des einführenden Beispiels

Der Schaltplan einer Baugruppe zeigt nun die einzelnen Bauelemente in symbolischer Darstellung und ihre Verbindungen. In der Programmierung ist dies der Programmablaufplan, der die einzelnen Befehle zeichnerisch darstellt.
Bild 62 zeigt den Programmablaufplan des einführenden Beispiels.

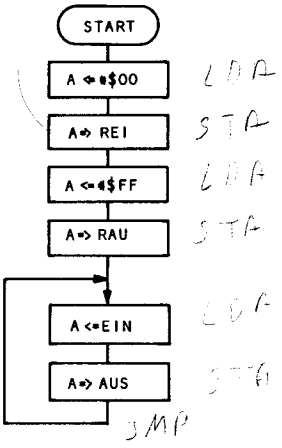


Bild 62: Programmablaufplan des einführenden Beispiels

Anhand des Schaltplans wird die Schaltung aus einzelnen Bauelementen aufgebaut; in der Programmierung codiert man die einzelnen Befehle in der Sprache des verwendeten Computers.
Bild 63 zeigt das symbolische Programm des einführenden Beispiels, Bild 64 links die binäre und rechts die verkürzte hexadezimale Schreibweise.

Der Aufbau eines größeren Gerätes aus einzelnen Baugruppen bietet den Vorteil, daß die Moduln austauschbar sind und einzeln entwickelt und getestet werden können. In der Programmierung heißen diese Programm-Moduln Unterprogramme.

Name	Befehl	Operand	Bemerkung
1	7	13	19 25 30 35
START	LDA	#$00	
	STA	REI	
	LDA	#$FF	
	STA	RAU	
SUSI	LDA	EIN	
	STA	AUS	
	JMP	SUSI	
	.END		

Bild 63: Symbolisches Programm (KIM)

Programmspeicher binär		hexadezimal	
Adresse	Inhalt	Adresse	Inhalt
0000000000000000	10101001	0000	A9
0000000000000001	00000000	0001	00
0000000000000010	10001101	0002	8D
0000000000000011	00000011	0003	03
0000000000000100	00010111	0004	17
0000000000000101	10101001	0005	A9
0000000000000110	11111111	0006	FF
0000000000000111	10001101	0007	8D
0000000000001000	00000001	0008	01
0000000000001001	00010111	0009	17
0000000000001010	10101101	000A	AD
0000000000001011	00000010	000B	02
0000000000001100	00010111	000C	17
0000000000001101	10001101	000D	8D
0000000000001110	00000000	000E	00
0000000000001111	00010111	000F	17
0000000000010000	01001100	0010	4C
0000000000010001	00001010	0011	0A
0000000000010010	00000000	0012	00

Bild 64: Binäres und hexadezimales Programm (KIM)

Bei der Entwicklung eines Gerätes werden zunächst die einzelnen Baugruppen, später ein Versuchsmuster unter allen Betriebsbedingungen geprüft. Bei der Programmentwicklung ist dies die Testphase, für die die alten Programmierersprüche gelten:

Durch Testen läßt sich nur die Anwesenheit, nicht aber die Abwesenheit von Fehlern nachweisen.

Ein Programm ohne Fehler gibt es nicht; sollte ein Programm beim Testen keinen Fehler zeigen, so ist dieser besonders heimtückisch. In der Praxis gibt es mehr Fehler zwischen Start und Stop, als ein Programmierer sich beim Testen ausdenken kann.

Bild 65 vergleicht nun die Programmierung mit der Entwicklung eines Gerätes.

Die folgenden Abschnitte beschreiben nun anhand einfacher Aufgaben

- Regeln zur symbolischen Darstellung von Befehlen,
- die Wirkung einzelner Befehle,
- die symbolische Schreibweise der Befehle,
- ihre symbolische Darstellung im Programmablaufplan,

Programm	Gerät
Systemanalyse	Aufgaben der Gesamtanlage
Problemanalyse	Aufgaben des Einzelgerätes
Programmblockplan	Blockschaltplan
Programmablaufplan	Schaltplan
Unterprogramm	Baugruppe
Befehl symbolisch	Bauelement im Schaltplan
Befehl binär im ROM	Bauelement auf Platine

Bild 65: Vergleich Programm - Gerät

- die symbolische Darstellung von Teilaufgaben im Programmblockplan und
- die wichtigsten Programmierverfahren.

Alle Beispiele können mit einem einfachen Bausatz nach Abschnitt 2.1. bis 2.3. und einer Zusatzschaltung nach Bild 48 getestet werden.

2.5 Das Programmieren in Assemblerschreibweise

In der Assemblerschreibweise werden die Befehle durch festgelegte Symbole und die Adressen durch frei wählbare Namen bezeichnet. Beispiele dafür zeigen die Bilder 52, 56 und 60. Für die Übersetzung in die eigentliche binäre oder in die kürzere hexadezimale Darstellung gibt es drei Möglichkeiten:

a. Der Programmierer übersetzt das Programm selbst, indem er die hexadezimalen Codes in Befehls- und Adreßlisten sucht. Dies wird immer dann bevorzugt, wenn nur ein einfacher Bausatz mit Hexadezimaltastatur zur Verfügung steht.

b. Mikrocomputer-Entwicklungssysteme enthalten residente Assembler. Dies sind Übersetzungsprogramme, die das symbolische Programm in die binäre Darstellung überführen und im Benutzerbereich in einem RAM ablegen, sodaß es anschließend getestet werden kann. Entwicklungssysteme enthalten häufig einen Editor, mit dessen Hilfe der Benutzer das symbolische Programm leicht eingeben und ändern kann. Hilfsprogramme unterstützen den Test der Benutzerprogramme und sorgen für die Übertragung des ausgetesteten Programms in Nur-Lese-Speicher (ROMs).

c. Steht ein Großrechner zur Verfügung, so kann ein Cross-Assembler das symbolische Programm übersetzen. Ein Simulationsprogramm testet dann das übersetzte Benutzerprogramm, indem es den Mikrocomputer auf dem Großrechner nachbildet.

Die Regeln, nach denen ein Assemblerprogramm aufgebaut werden muß, bilden die Assembler-Sprache. Man unterscheidet:

a. Assembler-Anweisungen, die dem Assembler Anweisungen zur Steuerung des Übersetzungsvorganges geben und nicht in Befehle übersetzt werden.

b. Daten-Definitionen, die Speicherplätze reservieren oder mit Konstanten füllen.

c. Assembler-Befehle, die in Maschinenbefehle von ein, zwei oder drei byte Länge übersetzt werden. Ein Assemblerbefehl ergibt einen Maschinenbefehl.

d. Makro-Befehle, das sind vom Programmierer vereinbarte Befehlsfolgen, die mit einem Namen aufgerufen werden und mehrere Maschinenbefehle erzeugen.

Ein Mikrocomputer besteht aus

a. dem Mikroprozessor, der die Befehle ausführt und die Daten verarbeitet.

b. Den E/A-Bausteinen, die von außen Daten aufnehmen und Daten nach außen abgeben.

c. Dem Datenspeicher für Zwischenergebnisse und variable Daten (RAM = Schreib-Lese-Speicher).

d. Dem Programmspeicher, der in einem ROM (Nur-Lese-Speicher) Konstanten und Befehle enthält. Beide sind wieder voneinander zu trennen, damit sichergestellt ist, daß nur Befehle in das Steuerwerk zur Ausführung gelangen.

Die Adressen der Bausteine liegen durch die Verdrahtung der Anschlüsse am Adreßbus fest. Vor der Programmierung müssen die hexadezimalen Adressen der E/A-Schnittstellen, der RAM-Variablenspeicher und der ROM-Programmspeicher festgelegt werden. Bei den Übungs- und Entwicklungsbausätzen ist dies bereits durch den Hersteller geschehen.
Bild 66 zeigt allgemein den Aufbau eines Assemblerprogramms, das diese Einteilung bereits berücksichtigt.

Bild 66: Aufbau eines Assemblerprogramms

Bei der Arbeit mit einem Entwicklungssystem legt der Benutzer sein Programm zunächst in einem RAM - Bereich ab, muß aber immer

daran denken, daß beim fertigen Gerät die Konstanten und Befehle in einem ROM liegen.

Bild 67 zeigt einen Vordruck, auf dem das Programm entworfen werden kann, bevor es übersetzt wird.

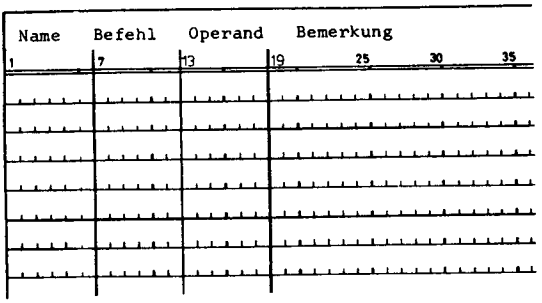


Bild 67: Vordruck zum Entwerfen eines Assemblerprogramms

Namen beginnen in Spalte 1. Sie bestehen aus höchstens sechs Zeichen und bezeichnen ein bestimmtes Datenbyte (Konstante, Variable, E/A-Register) oder einen Befehl (Sprungmarke).
Befehle beginnen in Spalte 7. Sie bestehen aus festgelegten Kennwörtern.
Operanden beginnen in Spalte 13. Dies können sein
- symbolische Namen von Registern, Daten oder Befehlen,
- dezimale Zahlenkonstanten,
- hexadezimale Zahlenkonstanten,
- Textkonstanten oder
- Adreßkonstanten.

Bemerkungen beginnen ab Spalte 19. Sie beschreiben das Programm in normaler Umgangssprache und werden nicht übersetzt.
Die Regeln der Assembler-Sprache werden von demjenigen aufgestellt, der das Übersetzungsprogramm schreibt. Leider gibt es für den Assembler keine einheitlichen Normen wie z.B. für die Programmiersprachen ALGOL und FORTRAN. Jeder Assembler, der gerade in einem bestimmten Entwicklungssystem oder Großrechner vorhanden ist, kann seine bestimmten von anderen Assemblern desselben Mikroprozessors abweichenden Regeln haben.

2.5.1 Eine vereinfachte Assemblersprache für den KIM-1 KIM

Allen Hexadezimalzahlen wird das Zeichen $ vorangestellt. Der Adreßzähler des Assemblers weist jedem Byte eine hexadezimale Adresse zu. Er kann durch die Assembler-Anweisung

```
*=$hhhh
```

auf einen Anfangswert hhhh gesetzt werden, dabei ist hhhh eine vierstellige Hexadezimalzahl. * ist das Symbol für den Adreßzähler.

Beispiel: Der Adreßzähler bekommt den Anfangswert 0100

```
|7
*  = $0100
```

Es ist sinnvoll, den Variablenbereich vom Programmbereich durch verschiedene Adreßzähler zu trennen, da sie später in verschiedenen Bauelementen liegen.

Durch die Assembleranweisung

```
Name     =$hhhh
```

ist es möglich, für symbolische Adressen, die in dem zu übersetzenden Programmteil nicht vorkommen, hexadezimale Adressen zu vereinbaren. Dies gilt besonders für die Adressen der Ein/Ausgabe-Register.

Beispiel: Das Eingaberegister mit der hexadezimalen Adresse 1702 erhält den Namen EIN

```
|1      |7
EIN     |=$1702
```

Die Assembleranweisung

```
.END
```

beendet den Übersetzungsvorgang.

Durch die Variablenreservierung

```
Name    *=*+Anzahl
```

wird eine beliebige Anzahl von Bytes im Variablenspeicher reserviert. Der Adreßzähler zählt um die angegebene Zahl von Bytes weiter, ohne jedoch Werte abzulegen.

Beispiel: Unter dem Namen SUSI sollen drei Bytes reserviert werden.

```
|1      |7
SUSI    |*=*+3
```

Durch die Konstantenvereinbarung

```
Name    .BYTE   Wert
```

wird der im Operandenteil angegebene Wert als hexadezimale Konstante von einem byte Länge abgelegt.

Beispiel: Unter dem Namen MAX soll die Konstante FF abgelegt werden

```
|1      |7       |13
|MAX    |.BYTE   |$FF
```

Durch die Konstantenvereinbarung

```
Name    .WORD  Wert
```

wird der im Operandenteil abgelegte Wert als hexadezimale Konstante von zwei byte Länge abgelegt. Als Wert dient meist eine symbolische Adresse, die als Adreßkonstante zwei Bytes benötigt.

Beispiel: Unter dem Namen OTTO soll die Adresse von MAX abgelegt werden.

```
|1      |7       |13
|OTTO   |.WORD   |MAX
```

Die Assembler-Befehle werden vom Übersetzungsprogramm in Maschinenbefehle von ein, zwei oder drei byte Länge übersetzt. Die Kennwörter der Befehle stimmen mit den Bezeichnungen des Prozessor-Herstellers überein.

Für alle Programmbeispiele gilt folgende Speicheraufteilung:

0000 bis 00EE : Variablen-RAM

0200 bis 02FF : Konstanten-ROM

0300 bis 03FF : Programm-ROM

Die Startadresse der Programme ist 0300.
Bild 51 enthält die Adressen der Ein/Ausgabe-Register.

2.5.2 Eine vereinfachte Assemblersprache für den Motorola 6800

6800

Allen Hexadezimalzahlen wird das Zeichen $ vorangestellt.
Die Assembleranweisung ORG (ORiGin = Ursprung) veranlaßt den Assembler, die folgenden Daten bzw. Befehle von einer bestimmten hexadezimalen Anfangsadresse ab fortlaufend anzuordnen. Dies geschieht durch einen Adreßzähler, der jedem Byte eine hexadezimale Adresse zuweist. Es ist sinnvoll, den Variablenbereich vom Programmbereich durch verschiedene Adreßzähler zu trennen, da sie später in verschiedenen Bauelementen liegen.

Allgemeine Form

```
        ORG   Zahl
```

Beispiel: Alle folgenden Bytes werden von der Adresse 0100 fortlaufend durchnumeriert

```
|7       |13
|ORG     |$0100
```

Durch die Assembleranweisung EQU (EQUal = gleich) ist es möglich, für symbolische Adressen, die in dem zu übersetzenden Programmteil nicht vorkommen, beliebige hexadezimale Adressen zu vereinbaren. Dies gilt besonders für die Adressen der Ein/Ausgabe-Register.

Allgemeine Form

```
| Name    EQU   Wert |
```

Beispiel: Das Eingaberegister mit der hexadezimalen Adresse 8000 erhält den symbolischen Namen EIN

```
|1       |7       |13
|EIN     |EQU     |$8000
```

Die Assembleranweisung END steht am Ende des Programms. Sie beendet die Übersetzung.

Allgemeine Form

```
|    END    |
```

Die Variablenreservierung RMB (Reserve Memory Bytes = reserviere Speicher Bytes) reserviert eine beliebige Anzahl von Bytes im Variablenbereich, ohne jedoch Werte abzulegen.

Allgemeine Form

```
| Name    RMB   Zahl |
```

Beispiel: Unter dem Namen SUSI sollen drei Bytes reserviert werden

```
|1       |7       |13
|SUSI    |RMB     |3
```

Die Konstantenvereinbarung FCB (Form Constant Byte = bilde Konstantenbyte) legt eine zweistellige hexadezimale Konstante von einem byte Länge ab.

Allgemeine Form

```
| Name    FCB   Wert |
```

Beispiel: Unter dem Namen MAX soll die Konstante FF abgelegt werden

```
|1      |7      |13
|MAX    |FCB    |$FF
```

Die Konstantenvereinbarung FDB (Form Double Constant Byte = bilde Doppelbytekonstante) legt eine vierstellige hexadezimale Konstante von zwei byte Länge ab.

Allgemeine Form

```
Name    FDB    Wert
```

Als Wert dient meist eine symbolische Adresse, die als Adreßkonstante zwei Bytes benötigt.

Beispiel: Unter dem Namen OTTO soll die Adresse von MAX abgelegt werden

```
|1      |7      |13
|OTTO   |FDB    |MAX
```

Die Assembler-Befehle werden vom Übersetzungsprogramm in Maschinenbefehle von ein, zwei oder drei byte Länge übersetzt. Die Kennwörter der Befehle stimmen mit den Bezeichnungen des Prozessor-Herstellers überein.

Für alle Programmbeispiele gilt folgende Speicheraufteilung:

0000 bis 00FF : Variablen-RAM

0100 bis 017F : Konstanten-ROM

0180 bis 01FF : Programm-ROM

Die Startadresse der Programme ist 0180.
Bild 55 enthält die Adressen der Ein/Ausgabe-Register.

2.5.3 Eine vereinfachte Assembler-Sprache für den 8080/8085 8080

Alle Hexadezimalzahlen müssen mit einer Ziffer beginnen und mit dem zusätzlichen Buchstaben H enden.

Beispiel: Die Hexadezimalzahl FF lautet in der Assemblerschreibweise 0FFH

Die Assembleranweisung ORG (ORiGin = Ursprung) veranlaßt den Assembler, die folgenden Daten bzw. Befehle von einer bestimmten Anfangsadresse ab fortlaufend anzuordnen. Dies geschieht durch einen Adreßzähler, der jedem Byte eine hexadezimale Adresse zuweist. Es ist sinnvoll, den Variablenbereich vom Programmbereich

durch verschiedene Adreßzähler zu trennen, da sie später in verschiedenen Bauelementen liegen.

Allgemeine Form

```
        ORG    Zahl
```

Beispiel: Alle folgenden Bytes werden von der Adresse 1200 fortlaufend durchnumeriert

```
|7      |13
|ORG    |1200H
```

Durch die Assembleranweisung EQU (EQUal = gleich) ist es möglich, für symbolische Adressen, die in dem zu übersetzenden Programmteil nicht vorkommen, beliebige hexadezimale Adressen zu vereinbaren.

Allgemeine Form

```
Name    EQU    Wert
```

Beispiel: Der symbolische Name PAUL erhält den hexadezimalen Wert FF

```
|1      |7      |13
|PAUL   |EQU    |0FFH
```

Die Assembleranweisung END steht am Ende des Programms. Sie beendet die Übersetzung.

Allgemeine Form

```
        END
```

Die Variablenreservierung DS (Definiere Speicher) reserviert eine beliebige Anzahl von Bytes im Variablenbereich, ohne jedoch Werte abzulegen.

Allgemeine Form

```
Name    DS     Zahl
```

Beispiel: Unter dem Namen SUSI sollen drei Bytes reserviert werden

```
|1      |7      |13
|SUSI   |DS     |3
```

Die Konstantenvereinbarung DB (Definiere Byte) legt eine zweistellige hexadezimale Konstante von einem byte Länge ab.

Allgemeine Form

```
Name    DB    Wert
```

Beispiel: Unter dem Namen MAX soll die Konstante FF abgelegt werden.

```
1       7     13
MAX     DB    0FFH
```

Die Konstantenvereinbarung DW (Definiere Wort) legt eine vierstellige hexadezimale Konstante von zwei byte Länge ab.

Allgemeine Form

```
Name    DW    Wert
```

Als Wert dient meist eine symbolische Adresse, die als Adreßkonstante zwei Bytes benötigt.

Beispiel: Unter dem Namen OTTO soll die Adresse von MAX abgelegt werden.

```
1       7     13
OTTO    DW    MAX
```

Die Assembler-Befehle werden vom Übersetzungsprogramm in Maschinenbefehle von ein, zwei oder drei byte Länge übersetzt. Die Kennwörter der Befehle stimmen mit den Bezeichnungen des Prozessor-Herstellers überein.

Für alle Programmbeispiele gilt folgende Speicheraufteilung:

2000 bis 20C7 : Variablen-RAM

2800 bis 287F : Konstanten-ROM

2880 bis 28FF : Programm-ROM

Die Startadresse der Programme ist 2880.
Bild 59b enthält die Adressen der Ein/Ausgabe-Register.

3. Der Datentransport

A

Transportbefehle transportieren Daten zwischen verschiedenen Speicherstellen des Mikrocomputers. Man unterscheidet:
- Register-Register-Operationen zwischen Registern des Mikro-Prozessors,
- Register-Speicher-Operationen (Speichern der Registerinhalte) und
- Speicher-Register-Operationen (Laden der Register).

Ein direkter Transport von Daten zwischen zwei Datenspeicherstellen ist nicht möglich, es ist immer ein Register des Prozessors als Zwischenspeicher zu benutzen. Die Befehlslisten enthalten einen Hinweis, ob das Bedingungsregister durch einen Transportbefehl beeinflußt wird. Bei allen Transportbefehlen behält die abgebende Speicherstelle ihren alten Wert, die aufnehmende Speicherstelle ändert sich.

3.1 Der Datentransport zwischen Registern des Prozessors

Bei allen Befehlen, die Daten zwischen den Registern des Mikroprozessors transportieren, enthält der 8-Bit-Funktionscode des Befehls bereits die internen Adressen der beteiligten Register. Es sind also 1-Byte-Befehle mit code-eigener Adressierung.

3.1.1 Die Register des KIM-1

KIM

Bild 68 zeigt die dem Programmierer zugänglichen Register des Prozessors und die Register-Register-Operationen.

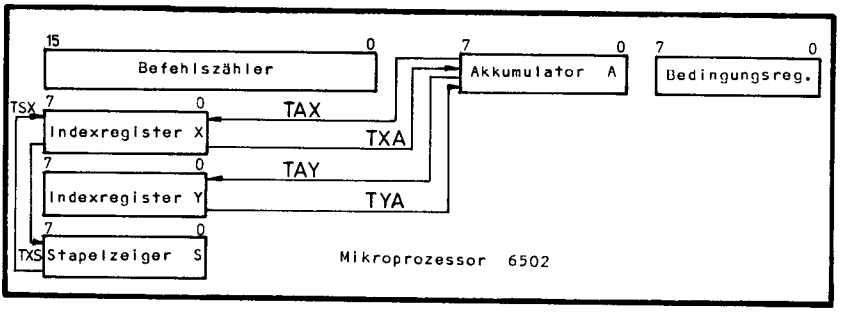


Bild 68: Die Register des KIM-1

Die folgende Tabelle faßt die Register-Register-Operationen zusammen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	V	B	D	I	Z	C	
TAX		code-eigen	AA	1	2	X					X		Transportiere Akkumulator nach X-Register
TXA		code-eigen	8A	1	2	X					X		Transportiere X-Register nach Akkumulator
TAY		code-eigen	A8	1	2	X					X		Transportiere Akkumulator nach Y-Register
TYA		code-eigen	98	1	2	X					X		Transportiere Y-Register nach Akkumulator
TXS		code-eigen	9A	1	2	X					X		Transportiere X-Register nach Stapelzeiger
TSX		code-eigen	BA	1	2	X					X		Transportiere Stapelzeiger nach X-Register

Der Buchstabe T ist eine Abkürzung für "Transfer = Transportiere". TAX bedeutet also "Transportiere den Inhalt des Akkumulators zum X-Register".

Die Spalte Befehl enthält den symbolischen Assemblerbefehl, die Spalte Hexa die entsprechende hexadezimale Codierung. In der Spalte Bedingungen zeigen die Kreuze in den Spalten N und Z, daß das Negativ- und das Zero-Bit je nach dem Inhalt der transportierten Daten beeinflußt wird. Bei einem Transport des Datenbytes FF wird das N-Bit auf 1 und das Z-Bit auf 0 gesetzt, denn FF = 11111111 ist negativ und nicht Null.

Für den Transport von Daten zwischen Registern, die nicht direkt miteinander verbunden sind, sind zwei Befehle erforderlich.

Beispiel: Laden des Akkumulators mit dem Stapelzeiger

```
   Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl   Operand

                        *=$0300
0300    BA      START   TSX
0301    8A              TXA
```

Das Beispielprogramm enthält rechts die symbolische und links die hexadezimale Schreibweise der beiden Befehle. Die erste Anweisung setzt den Adreßzähler auf 0300. Der erste Befehl TSX bringt den Stapelzeiger in das X-Register. Der zweite Befehl TXA bringt das X-Register in den Akkumulator.

Das folgende Beispiel zeigt, wie man den Stapelzeiger mit einem Anfangswert laden kann, der sich im Akkumulator befindet.

```
   Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl   Operand

                        *=$0300
0300    AA      START   TAX
0301    9A              TXS
```

In beiden Beispielen dient das X-Register als Zwischenspeicher; es wird dabei zerstört.

3.1.2 Die Register des Motorola 6800

Bild 69 zeigt die dem Programmierer zugänglichen Register des Prozessors und die Register-Register-Operationen.

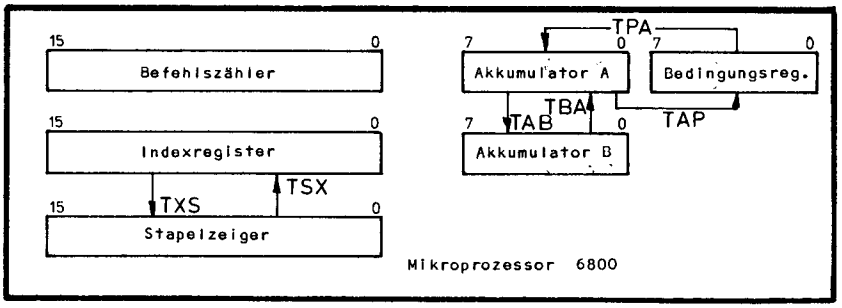


Bild 69: Die Register des 6800

Die folgende Tabelle faßt die Register-Register-Operationen zusammen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen -|-|H|I|N|Z|V|C	
TAB		code-eigen	16	1	2	XXO	Transportiere Akku A nach Akku B
TBA		code-eigen	17	1	2	XXO	Transportiere Akku B nach Akku A
TAP		code-eigen	06	1	2	wie Akku	Transportiere Akku A nach Bedingungsreg.
TPA		code-eigen	07	1	2		Transportiere Bedingungsreg. nach Akku A
TXS		code-eigen	35	1	4		Transportiere X-Register (-1) nach Stapelz.
TSX		code-eigen	30	1	4		Transportiere Stapelz. (+1) nach X-Register

Der Buchstabe T ist eine Abkürzung für "Transfer = Transportiere". TAB bedeutet also "Transportiere Akkumulator A zum Akkumulator B".
Die Spalte Befehl enthält den symbolischen Assemblerbefehl, die Spalte Hexa die entsprechende hexadezimale Codierung. In der Spalte Bedingungen zeigen Kreuze in den Spalten N und Z, daß das Negativ- und das Zero-Bit je nach dem Inhalt der transportierten Daten beeinflußt wird. Das V-Bit (Überlauf) wird in jedem Fall Null gesetzt. Bei einem Transport des Datenbytes FF zwischen den beiden Akkumulatoren wird das N-Bit auf 1 und das Z-Bit auf 0 gesetzt, denn FF = 11111111 ist negativ und nicht Null.

Für den Transport von Daten zwischen Registern, die nicht direkt miteinander verbunden sind, sind zwei Befehle erforderlich.

Beispiel: Laden des Akkumulators B mit dem Bedingungsregister

```
   Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl   Operand

                        ORG      $0180
0180    07      START   TPA
0181    16              TAB
```

MOV ist eine Abkürzung für MOVe = bewegen. MOV A,B bedeutet also "Bewege B-Register nach A-Register". Besser ist der Ausdruck "Lade A-Register mit B-Register".

Lade Register	mit A	mit B	mit C	mit D	mit E	mit H	mit L
A	MOV A,A 7F	MOV A,B 78	MOV A,C 79	MOV A,D 7A	MOV A,E 7B	MOV A,H 7C	MOV A,L 7D
B	MOV B,A 47	MOV B,B 40	MOV B,C 41	MOV B,D 42	MOV B,E 43	MOV B,H 44	MOV B,L 45
C	MOV C,A 4F	MOV C,B 48	MOV C,C 49	MOV C,D 4A	MOV C,E 4B	MOV C,H 4C	MOV C,L 4D
D	MOV D,A 57	MOV D,B 50	MOV D,C 51	MOV D,D 52	MOV D,E 53	MOV D,H 54	MOV D,L 55
E	MOV E,A 5F	MOV E,B 58	MOV E,C 59	MOV E,D 5A	MOV E,E 5B	MOV E,H 5C	MOV E,L 5D
H	MOV H,A 67	MOV H,B 60	MOV H,C 61	MOV H,D 62	MOV H,E 63	MOV H,H 64	MOV H,L 65
L	MOV L,A 6F	MOV L,B 68	MOV L,C 69	MOV L,D 6A	MOV L,E 6B	MOV L,H 6C	MOV L,L 6D

Bild 71: Tabelle der MOV-Befehle

Beispiel: Lade die Register B und C mit den Registern H und L

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	2880H
2880	44	START	MOV	B,H
2881	4D		MOV	C,L

Das Beispielprogramm enthält rechts die symbolische und links die hexadezimale Schreibweise der Befehle.
Die erste Anweisung setzt den Adreßzähler auf 2880.
Der erste Befehl lädt das B-Register mit dem Inhalt des H-Registers. Die symbolische Adresse (Name) START entspricht der hexadezimalen Adresse 2880. Der symbolische Befehl MOV B,H lautet hexadezimal 44.

Während der MOV-Befehl die Register als Einzelregister von acht bit Länge behandelt, transportieren die folgenden drei Befehle die 16 Bits des Registerpaares H,L in ein anderes 16-Bit-Register.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C		
XCHG		code-eigen	EB	1	4			Vertausche Registerpaare D,E und H,L
SPHL		code-eigen	F9	1	5			Lade Stapelzeiger mit Registerpaar H,L
PCHL		code-eigen	E9	1	5			Lade Befehlszähler mit Registerpaar H,L

XCHG bedeutet eXCHanGe = vertausche.
SPHL bedeutet Lade den Stack Pointer = Stapelzeiger mit dem H, L - Register.
PCHL bedeutet Lade den Program Counter = Befehlszähler mit dem H, L - Register.

Beispiel: Lade den Stapelzeiger mit dem Registerpaar H,L

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	2880H
2880	F9	START	SPHL	

3.2 Das Laden von Registern A

Bild 72 zeigt das Laden eines Registers aus dem Datenspeicher. Dabei kann es sich um eine Variable (RAM), eine Konstante (ROM) oder um einen Wert aus einem Eingaberegister handeln.

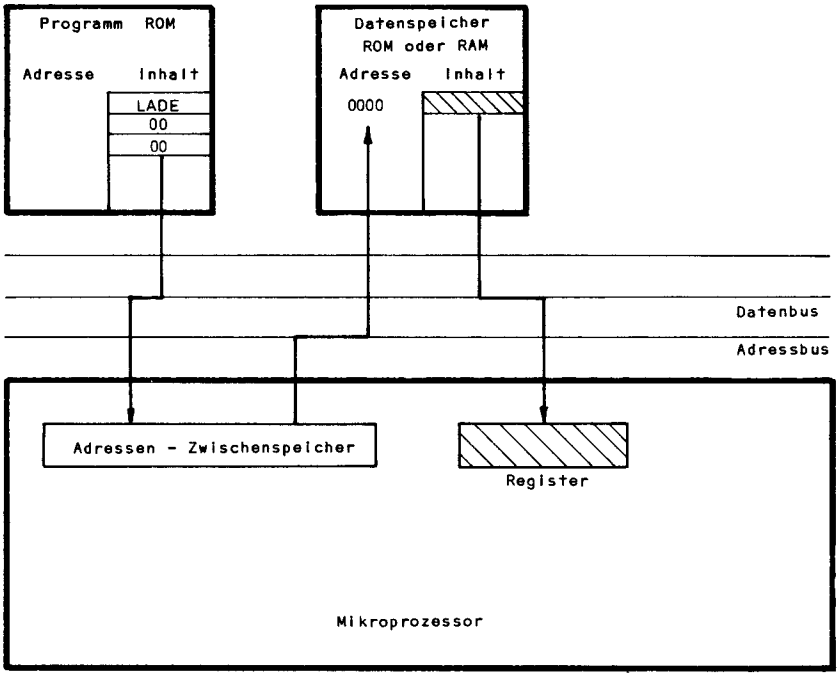


Bild 72: Laden eines Registers mit einer Speicherstelle

Der Ladebefehl besteht aus dem Funktionscode LADE und einer Adresse, die in dem Beispiel willkürlich 0000 gesetzt wurde. Er ist damit ein 3-Byte-Befehl. Bei der Ausführung des Befehls gelangt der Funktionscode in das Befehlsregister und die Adresse in einen Zwischenspeicher, von dem aus sie über den Adreßbus in den Datenspeicher übertragen wird. Der Inhalt der adressierten Speicherstelle wird dann über den Datenbus in das Register übertragen. Diese Adressierung wird absolut genannt.

Ein häufiger Sonderfall ist das Laden eines Registers mit einer einzelnen Konstanten. Dabei ist es einfacher, die Konstante anstelle der Adresse unmittelbar hinter dem Funktionscode abzulegen und direkt in das Register zu bringen.

Bild 73 zeigt das Laden eines Registers mit einer Konstanten, die unmittelbar hinter dem Funktionscode folgt. Diese Art der Adressierung nennt man unmittelbar.

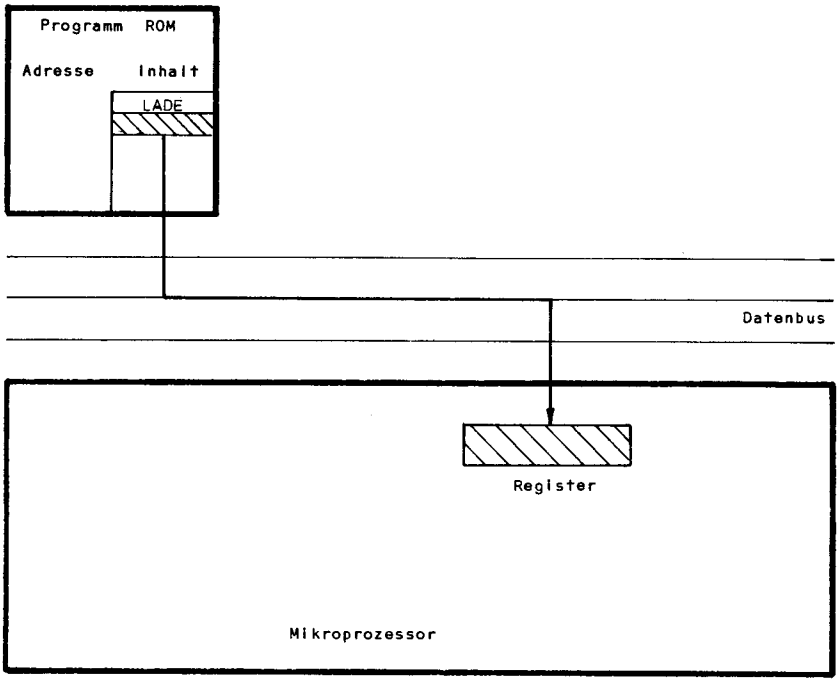


Bild 73: Laden eines Registers mit einer Konstanten

Der Befehl besteht aus dem Funktionscode und einer Konstanten, die unmittelbar hinter ihm folgt. Bei der Ausführung des Befehls gelangt der Funktionscode in das Befehlsregister, die Konstante direkt in das Register.

3.2.1 Das Laden der KIM-1 Register KIM

Der 16 bit breite Adreßbus des KIM-1 erfordert zur Adressierung im gesamten Speicherbereich zwei Bytes, die hinter dem Funktionscode abgelegt werden (3-Byte-Befehl). Beim KIM-1 werden die wertniedrigsten Bits der Adresse im zweiten Byte und die werthöchsten im dritten Byte abgelegt. In der Assemblerschreibweise enthält das Operandenfeld die symbolische Adresse der Datenspeicherstelle.

Beispiel: Lade den Akkumulator mit dem Inhalt der Speicherstelle EIN; hexadezimal 1702.

```
Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl   Operand

                EIN     =$1702
                        *=$0300
0300    AD      START   LDA      EIN
        02
        17
```

Die folgenden Befehle laden drei Register des Prozessors mit dem Inhalt einer Speicherstelle in der absoluten Adressierung; d.h. die Adresse kann in dem gesamten Speicherbereich liegen.
Ein X in der Spalte Bedingungen bedeutet, daß das entsprechende Bit des Bedingungsregisters je nach Wert des transportierten Datenbytes gesetzt wird.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
LDA	Adresse	absolut	AD	3	4	X				X		Lade Akkumulator (absolut adressiert)
LDX	Adresse	absolut	AE	3	4	X				X		Lade X-Register (absolut adressiert)
LDY	Adresse	absolut	AC	3	4	X				X		Lade Y-Register (absolut adressiert)

Die Buchstaben LD sind eine Abkürzung für LoaD = Lade.
LDA bedeutet also "LaDe den Akkumulator".
Liegt die Speicheradresse im Bereich von 0000 bis 00FF, so ist das werthöchste Byte der Adresse 00. Unter dieser Voraussetzung kann man die Adresse um das Byte 00 verkürzen, muß dies allerdings im Funktionscode vermerken. Bei der Ausführung des Befehls fügt der Prozessor das unterdrückte Byte 00 der Adresse automatisch hinzu.
Legt man also häufig gebrauchte Daten oder die Adressen der Ein/Ausgabe-Register in den Bereich 0000 bis 00FF, so kann man mit dieser zero-page (Seite Null) genannten Adressierung Speicherplatz und Ausführungszeit sparen. Der symbolische Befehl erhält hier in allen Beispielen den Zusatz Z. Im Operandenfeld steht die symbolische Adresse der Datenspeicherstelle. In den Befehlslisten des Herstellers fehlt der Buchstabe Z, der Assembler muß dann selbständig die zero-page Adressierung erkennen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
LDAZ	Adresse	zero-page	A5	2	3	X				X		Lade Akkumulator (zero-page adressiert)
LDXZ	Adresse	zero-page	A6	2	3	X				X		Lade X-Register (zero-page adressiert)
LDYZ	Adresse	zero-page	A4	2	3	X				X		Lade Y-Register (zero-page adressiert)

Beispiel: Lade den Akkumulator mit dem Inhalt der Speicherstelle EVA, die in Seite Null liegt.

```
Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl   Operand

                        *=$0000
0000            EVA     *=*+1

                        *=$0300
0300    A5      START   LDAZ     EVA
        00
```

Zum Laden eines Registers mit einer Konstanten kann diese im zweiten Byte des Befehls abgelegt werden. Der Operandenteil des Befehls enthält das Zeichen # und anschließend die Konstante.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	V	B	D	I	Z	C	
LDA	# konst	unmittelbar	A9	2	2	X					X		Lade Akkumulator mit Konstanten
LDX	# konst	unmittelbar	A2	2	2	X					X		Lade X-Register mit Konstanten
LDY	# konst	unmittelbar	A0	2	2	X					X		Lade Y-Register mit Konstanten

Beispiel: Lade den Akkumulator mit der Konstanten 00

```
   Hexadezimal          Symbolisch

Adresse Inhalt   Name    Befehl    Operand

                         *=$0300
0300    A9       START   LDA       #$00
        00
```

Der Stapelzeiger läßt sich nicht direkt, sondern nur über das X-Register laden. Das folgende Beispiel lädt den Stapelzeiger mit der Konstanten FF.Beim KIM-1 liegt der Stapel immer in Seite 1, d.h. der Prozessor setzt bei der Adressierung des Stapels das werthöhere Byte der Stapeladresse immer 01.

Beispiel: Laden des Stapelzeigers mit der Anfangsadresse FF

```
   Hexadezimal          Symbolisch

Adresse Inhalt   Name    Befehl    Operand

                         *=$0300
0300    A2       START   LDX       #$FF
        FF
0302    9A               TXS
```

Die indizierte und die indirekte Adressierung, die sich ebenfalls auf die Ladebefehle anwenden lassen, werden später erklärt.

Das Bedingungsregister des KIM-1-Prozessors hat folgenden Aufbau:

7	6		4	3	2	1	0
N	V		B	D	I	Z	C

N = 0 : Vorzeichen positiv
N = 1 : Vorzeichen negativ
V = 0 : kein Zahlenüberlauf
V = 1 : Zahlenüberlauf
B = 0 : kein Interrupt durch den BRK-Befehl
B = 1 : Interrupt durch den BRK-Befehl
D = 0 : keine Dezimalarithmetik, also Dualarithmetik
D = 1 : Dezimalarithmetik eingeschaltet
I = 0 : Interrupt durch IRQ und BRK möglich
I = 1 : Interrupt durch IRQ und BRK gesperrt
Z = 0 : Ergebnis nicht Null
Z = 1 : Ergebnis Null
C = 0 : Borgen (Subtraktion) bzw. kein Übertrag (Addition)
C = 1 : Übertrag (Addition) bzw. kein Borgen (Subtraktion)

Ein Teil der Bedingungsbits läßt sich durch Befehle setzen und rücksetzen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B D I Z C	
CLV		code-eigen	B8	1	2	0	Überlaufanzeige gelöscht
CLD		code-eigen	D8	1	2	0	Dualarithmetik eingeschaltet
SED		code-eigen	F8	1	2	1	Dezimalarithmetik eingeschaltet
CLI		code-eigen	58	1	2	0	Interrupt-Sperre aufgehoben
SEI		code-eigen	78	1	2	1	Interrupt gesperrt
CLC		code-eigen	18	1	2	0	Carry-Bit gelöscht (vor Addition)
SEC		code-eigen	38	1	2	1	Carry-Bit gesetzt (vor Subtraktion)

Die Buchstaben CL sind eine Abkürzung für CLear = lösche. CLC bedeutet also "Lösche Carry-Bit".
Die Buchstaben SE sind eine Abkürzung für SEt = setze. SEC bedeutet also "Setze Carry-Bit".
Das N-Bit und das Z-Bit lassen sich nur durch eine Operation mit entsprechenden Daten beeinflussen.

3.2.2 Das Laden der 6800-Register **6800**

Der 16 bit breite Adreßbus des 6800 erfordert zur Adressierung im gesamten Speicherbereich zwei Bytes, die hinter dem Funktionscode abgelegt werden (3-Byte-Befehl). Diese Adressierungsart heißt hier absolut, in der Bezeichnung des Herstellers extended = ausgedehnt. Die werthöchsten Bits der Adresse werden im zweiten Byte und die wertniedrigsten im dritten Byte abgelegt. In der Assemblerschreibweise enthält das Operandenfeld die symbolische Adresse einer Daten-Speicherstelle.

Beispiel: Lade den Akkumulator A mit dem Inhalt der Datenspeicherstelle EIN = 8006.

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
		EIN	EQU	$8006
			ORG	$0180
0180	B6	START	LDA A	EIN
	80			
	06			

Die folgenden Befehle laden ein 8-Bit Register (Akkumulator A oder B) mit dem Inhalt der absolut adressierten Speicherstelle oder ein 16-Bit-Register (Indexregister oder Stapelzeiger) mit dem Inhalt der adressierten Speicherstelle und der folgenden, also mit zwei Bytes. Ein X in der Spalte Bedingungen zeigt an,

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
LDA A	Adresse	absolut	B6	3	4	X X 0	Lade Akkumulator A (extended adressiert)
LDA B	Adresse	absolut	F6	3	4	X X 0	Lade Akkumulator B (extended adressiert)
LDX	Adresse	absolut	FE	3	5	X X 0	Lade X-Register (extended adressiert)
LDS	Adresse	absolut	BE	3	5	X X 0	Lade Stapelzeiger (extended adressiert)

daß das entsprechende Bit je nach dem Wert des transportierten Bytes beeinflußt wird. Bei den Befehlen, die zwei Bytes transportieren, wird das N-Bit nur durch Bit 15 beeinflußt.

Die Buchstaben LD sind eine Abkürzung für LoaD = LaDe.
LDA A bedeutet also"LaDe den Akkumulator A".

Liegt die Speicheradresse im Bereich von 0000 bis 00FF, so ist das werthöchste Byte der Adresse 00. Unter dieser Voraussetzung kann man die Adresse um das Byte 00 verkürzen, muß dies allerdings im Funktionscode vermerken. Bei der Ausführung des Befehls wird das werthöhere Byte 00 automatisch wieder hinzugefügt. Legt man also häufig gebrauchte Daten oder die Adressen der Ein/Ausgabe-Register in den Bereich von 0000 bis 00FF, so kann man mit dieser zero-page (Seite Null) genannten Adressierung Speicherplatz und Ausführungszeit sparen, da der Befehl nur noch aus zwei Bytes besteht. Der symbolische Befehl erhält bei der zero-page Adressierung den Zusatz Z, das Operandenfeld nimmt den symbolischen Namen der Speicherstelle auf. In den Befehlslisten des Herstellers heißt diese Adressierung direct, der Zusatz Z fehlt. Es ist Aufgabe des Assemblers, die Adressierungsart zu erkennen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
LDAZ A	Adresse	zero-page	96	2	3	XXO	Lade Akkumulator A (direct adressiert)
LDAZ B	Adresse	zero-page	D6	2	3	XXO	Lade Akkumulator B (direct adressiert)
LDXZ	Adresse	zero-page	DE	2	4	XXO	Lade X-Register (direct adressiert)
LDSZ	Adresse	zero-page	9E	2	4	XXO	Lade Stapelzeiger (direct adressiert)

Beispiel: Lade den Akkumulator A mit dem Inhalt der Speicherstelle EVA , die in Seite Null liegt

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	$0010
0010		EVA	RMB	1
			ORG	$0180
0180	96	START	LDAZ A	EVA
	10			

Zum Laden eines Akkumulators mit einer Konstanten von einem byte Länge kann diese unmittelbar nach dem Funktionscode im zweiten Byte des Befehls abgelegt werden; bei einem 16-Bit-Register liegt die zwei byte lange Konstante im zweiten und dritten Byte. Der Operandenteil enthält das Zeichen # und danach die Konstante.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
LDA A	#konst	unmittelbar	86	2	2	XXO	Lade Akkumulator A mit 1-Byte-Konstanten
LDA B	#konst	unmittelbar	C6	2	2	XXO	Lade Akkumulator B mit 1-Byte-Konstanten
LDX	#konst	unmittelbar	CE	3	3	XXO	Lade X-Register mit 2-Byte-Konstanten
LDS	#konst	unmittelbar	8E	3	3	XXO	Lade Stapelzeiger mit 2-Byte-Konstanten

Beispiel: Lade den Akkumulator A mit der Konstanten 00

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	$0180
0180	86	START	LDA A	#$00
	00			

Zum Löschen von Registern und Speicherstellen können folgende Befehle verwendet werden.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
CLR A		code-eigen	4F	1	2	0 1 0 0	Lösche Akkumulator A mit binären Nullen
CLR B		code-eigen	5F	1	2	0 1 0 0	Lösche Akkumulator B mit binären Nullen
CLR	Adresse	absolut	7F	3	6	0 1 0 0	Lösche Speicherstelle mit binären Nullen

Die Zeichen CLR sind eine Abkürzung für CLeaR = lösche.

Das Bedingungsregister des 6800 hat folgenden Aufbau:

Bit	7	6	5	4	3	2	1	0
			H	I	N	Z	V	C

Bit 6 und 7 sind immer 1 !

H = 0 : kein Halb-Carry, kein Übertrag von Bit 3 nach Bit 4
H = 1 : Halb-Carry, Übertrag von Bit 3 nach Bit 4
I = 0 : keine Interruptsperre
I = 1 : Interruptsperre eingeschaltet
N = 0 : Vorzeichen positiv
N = 1 : Vorzeichen negativ
Z = 0 : Ergebnis nicht gleich Null
Z = 1 : Ergebnis gleich Null
V = 0 : kein Überlauf
V = 1 : Überlauf
C = 0 : Übertrag Null, kein Übertrag
C = 1 : Übertrag

Ein Teil der Bedingungsbits läßt sich durch folgende Befehle setzen oder rücksetzen:

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
CLI		code-eigen	0E	1	2	0	Interrupt-Sperre aufgehoben
SEI		code-eigen	0F	1	2	1	Interrupt gesperrt
CLV		code-eigen	0A	1	2	0	Überlaufanzeige gelöscht
SEV		code-eigen	0B	1	2	1	Überlaufanzeige gesetzt
CLC		code-eigen	0C	1	2	0	Carry-Bit gelöscht
SEC		code-eigen	0D	1	2	1	Carry-Bit gesetzt

Die Buchstaben CL sind eine Abkürzung für CLear = lösche. CLC bedeutet also "Lösche Carry-Bit". Die Buchstaben SE sind eine Abkürzung für SEt = setze. SEC bedeutet also "Setze Carry-Bit".
Außerdem ist es möglich, durch den Befehl TAP das Bedingungsregister mit dem Inhalt des Akkumulators A zu laden.

3.2.3 Das Laden der 8080/8085-Register

Der 16 bit breite Adreßbus erfordert zur Adressierung im gesamten Speicherbereich zwei Bytes, die hinter dem Funktionscode abgelegt werden (3-Byte-Befehl). Dabei stehen die wertniedrigsten Bits im zweiten Byte und die werthöchsten im dritten. In der Assemblerschreibweise enthält das Operandenfeld die symbolische Adresse einer Datenspeicherstelle.

Beispiel: Lade den Akkumulator mit dem Inhalt der Speicherstelle MAUS = 2800

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
		MAUS	EQU	2800H
			ORG	2880H
2880	3A	START	LDA	MAUS
	00			
	28			

Die in diesem Abschnitt gezeigte absolute Adressierung läßt sich nur auf den Akkumulator und das Registerpaar H,L anwenden. Zum Laden der anderen Register wird die Registerpaar-Adressierung verwendet. Dabei steht die Adresse der Datenspeicherstelle nicht im Befehl, sondern in einem Registerpaar. In den Unterlagen des Herstellers heißt die absolute Adressierung direkte Speicheradressierung.

Die folgenden Befehle laden den Akkumulator mit einem Byte bzw. das Registerpaar H,L mit zwei Bytes.Dabei gelangt die adressierte Speicherstelle in das L-Register, die folgende in das H-Register.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C		
LDA	Adresse	absolut	3A	3	13			Lade Akkumulator direkt
LHLD	Adresse	absolut	2A	3	16			Lade Registerpaar H,L direkt

LDA bedeutet "LoaD Accumulator = LaDe Akkumulator. LHLD bedeutet Load H,L-Register Direct = LaDe H,L-Register Direkt".

Die anderen Register können über Register-Register-Befehle oder über die Registerpaar-Adressierung geladen werden. Eine zero-page Adressierung gibt es nicht.

Zum Laden eines Registers mit einer Konstanten von einem byte Länge oder eines Registerpaares mit einer Konstanten von zwei byte Länge können diese Konstanten unmittelbar hinter dem Funktionscode abgelegt werden. In der Assemblerschreibweise enthält der Operandenteil die Registerbezeichnung und durch ein Komma getrennt die Konstante.

Die Abkürzung MVI bedeutet MoVe Immediate, frei übersetzt "Lade Register mit unmittelbar folgender Konstanten".
Die Abkürzung LXI bedeutet Load X-Register Immediate, frei übersetzt "Lade X-Register mit unmittelbar folgender Konstanten".

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
MVI	A,konst	unmittelbar	3E	2	7		Lade Akkumulator mit 1-Byte-Konstanten
MVI	B,konst	unmittelbar	06	2	7		Lade B-Register mit 1-Byte-Konstanten
MVI	C,konst	unmittelbar	0E	2	7		Lade C-Register mit 1-Byte-Konstanten
MVI	D,konst	unmittelbar	16	2	7		Lade D-Register mit 1-Byte-Konstanten
MVI	E,konst	unmittelbar	1E	2	7		Lade E-Register mit 1-Byte-Konstanten
MVI	H,konst	unmittelbar	26	2	7		Lade H-Register mit 1-Byte-Konstanten
MVI	L,konst	unmittelbar	2E	2	7		Lade L-Register mit 1-Byte-Konstanten
LXI	B,konst	unmittelbar	01	3	10		Lade Registerpaar B,C mit 2-Byte-Konst.
LXI	D,konst	unmittelbar	11	3	10		Lade Registerpaar D,E mit 2-Byte-Konst.
LXI	H,konst	unmittelbar	21	3	10		Lade Registerpaar H,L mit 2-Byte-Konst.
LXI	S,konst	unmittelbar	31	3	10		Lade Stapelzeiger mit 2-Byte-Konstanten

Beispiel: Lade den Akkumulator mit der Konstanten 00

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,00H
	00			

Das Bedingungsregister des 8080/8085 hat folgenden Aufbau:

Bit	7	6	5	4	3	2	1	0
	N	Z		H		P		C

Die Bitpositionen 1, 3 und 5 sind nicht bestimmt.

N = 0 : Vorzeichen positiv
N = 1 : Vorzeichen negativ
Z = 0 : Ergebnis nicht Null
Z = 1 : Ergebnis Null
H = 0 : kein Halb-Übertrag von Bit 3 nach Bit 4
H = 1 : Halb-Übertrag von Bit 3 nach Bit 4
P = 0 : Summe aller 1er Bits ungerade (Parität)
P = 1 : Summe aller 1er Bits gerade (Parität)
C = 0 : kein Übertrag
C = 1 : Übertrag

Nur das C-Bit läßt sich durch Befehle setzen oder negieren.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
STC		code-eigen	37	1	4	1	Setze das Carry-Bit auf 1
CMC		code-eigen	3F	1	4	C	Negiere das Carry-Bit

STC bedeutet SeT Carry = SeTze Carry-Bit.
CMC bedeutet CoMplement Carry = CoMplementiere Carry-Bit.

3.3 Das Speichern von Registern

Der Inhalt eines Registers kann nur in den Variablenbereich eines RAM-Speichers oder in ein Ausgaberegister gebracht werden. Das erste Byte eines Speicherbefehls enthält den Funktionscode, das zweite und dritte die Speicheradresse. Der Inhalt des abgebenden Registers bleibt erhalten, der Inhalt der aufnehmenden Speicherstelle wird mit dem neuen Wert überschrieben.
Bild 74 zeigt das Speichern eines Registers in die Speicherstelle 1111.

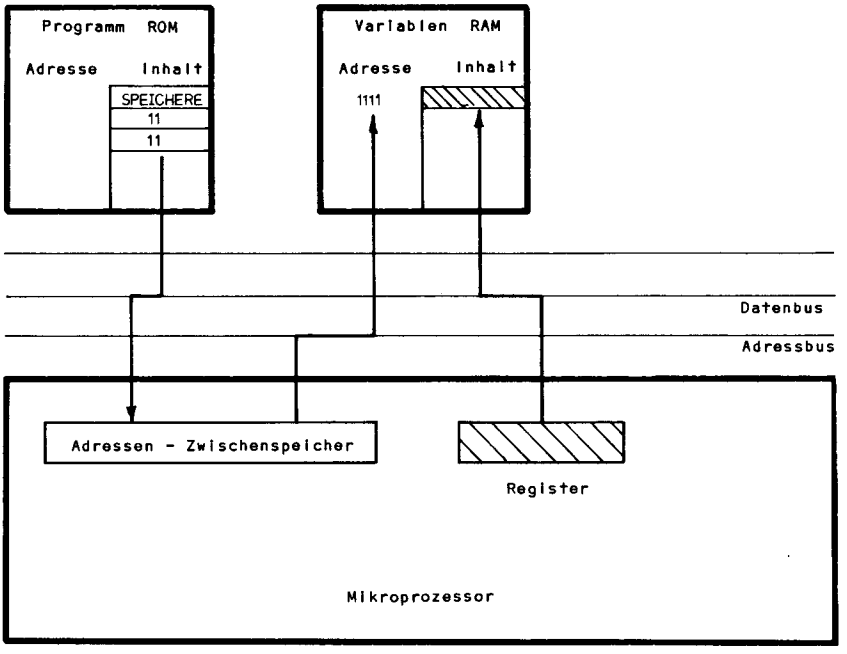


Bild 74: Speichern eines Registers

3.3.1 Das Speichern der KIM-1-Register KIM

Bei der absoluten Adressierung, die den gesamten Speicherbereich umfaßt, besteht die Adresse aus zwei Bytes. Die wertniedrigsten Bits stehen im zweiten Byte, die werthöchsten im dritten. Der gesamte Befehl besteht aus drei Bytes.

Die folgenden Befehle speichern den Inhalt eines Registers in eine Speicherstelle, deren Adresse in der Assemblerschreibweise im Operandenfeld steht. Das Bedingungsregister wird nicht beeinflußt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	
STA	Adresse	absolut	8D	3	4		Speichere Akkumulator (absolut adressiert)
STX	Adresse	absolut	8E	3	4		Speichere X-Register (absolut adressiert)
STY	Adresse	absolut	8C	3	4		Speichere Y-Register (absolut adressiert)

Die Buchstaben ST sind eine Abkürzung für STore = speichere. STA bedeutet also "Speichere Akkumulator".
Beispiel: Speichere den Inhalt des Akkumulators nach EVA

```
   Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl    Operand

                        *=$0010
0010            EVA     *=*+1

                        *=$0300
0300    8D      START   STA       EVA
        10
        00
```

Bei der zero-page Adressierung muß die Adresse im Bereich von 0000 bis 00FF liegen. Unter dieser Voraussetzung kann man die Adresse um ein Byte verkürzen, muß dies allerdings im Funktionscode vermerken. Der symbolische Befehl erhält hier (entgegen der Herstellerbezeichnung) den Zusatz Z, das Operandenfeld nimmt die symbolische Speicheradresse auf. Der hexadezimale Befehl besteht aus zwei Bytes. Das Bedingungsregister wird nicht beeinflußt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	
STAZ	Adresse	zero-page	85	2	3		Speichere Akkumulator (zero-page adress.)
STXZ	Adresse	zero-page	86	2	3		Speichere X-Register (zero-page adress.)
STYZ	Adresse	zero-page	84	2	3		Speichere Y-Register (zero-page adress.)

Beispiel: Speichere den Inhalt des Akkumulators nach EVA mit zero-page Adressierung

```
   Hexadezimal          Symbolisch

Adresse Inhalt  Name    Befehl    Operand

                        *=$0010
0010            EVA     *=*+1

                        *=$0300
0300    85      START   STAZ      EVA
        10
```

Beide Beispiele adressieren dieselbe Speicherstelle. Das erste in absoluter Adressierung mit einem 3-Byte-Befehl (Funktionscode 8D) und das zweite in zero-page Adressierung mit einem 2-Byte-Befehl (Funktionscode 85).

3.3.2 Das Speichern der 6800-Register

6800

Bei der absoluten Adressierung, die den gesamten Adreßbereich umfaßt, besteht der gesamte Befehl aus drei Bytes, die Adresse aus zwei Bytes. Die werthöchsten Bits stehen im zweiten Byte des Befehls, die wertniedrigsten im dritten. Die hier absolut genannte Adressierung heißt beim Hersteller des Mikroprozessors extended = ausgedehnt.

Die folgenden Befehle speichern ein Byte aus einem Akkumulator in die adressierte Speicherstelle bzw. zwei Bytes aus dem Indexregister oder Stapelzeiger in die adressierte und die darauf

folgende Speicherstelle. Ein X in der Spalte Bedingungen zeigt an, daß das entsprechende Bit des Bedingungsregisters durch die transportierten Daten verändert wird. Bei allen Befehlen, die zwei Bytes transportieren, wird das N-Bit durch Bit 15 beeinflußt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
STA A	Adresse	absolut	B7	3	5	X X 0	Speichere Akku A (extended adressiert)
STA B	Adresse	absolut	F7	3	5	X X 0	Speichere Akku B (extended adressiert)
STX	Adresse	absolut	FF	3	6	X X 0	Speichere X-Register (extended adressiert)
STS	Adresse	absolut	BF	3	6	X X 0	Speichere Stapelzeiger (extended addr.)

Die Buchstaben ST sind eine Abkürzung für STore = Speichere. STA A bedeutet also "Speichere den Akkumulator A".

Soll eine variable Speicherstelle einen bestimmten Wert bekommen, so ist zuerst ein Register mit dem Wert zu laden und dann in die Speicherstelle zu bringen.

Beispiel: Die Speicherstelle OTTO soll auf FF gesetzt werden

```
   Hexadezimal          Symbolisch
Adresse Inhalt   Name    Befehl    Operand

                         ORG       $0000
0000             OTTO    RMB       1

                         ORG       $0180
0180     86      START   LDA A     #$FF
         FF
0182     B7              STA A     OTTO
         00
         00
```

Bei der zero-page Adressierung muß die Adresse im Bereich von 0000 bis 00FF liegen. Unter dieser Voraussetzung kann man die Adresse um das erste Byte 00 verkürzen, muß dies jedoch im Funktionscode vermerken. Entgegen der Bezeichnung des Herstellers erhält hier der symbolische Befehl den Zusatz Z.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
STAZ A	Adresse	zero-page	97	2	4	X X 0	Speichere Akku A (direct adressiert)
STAZ B	Adresse	zero-page	D7	2	4	X X 0	Speichere Akku B (direct adressiert)
STXZ	Adresse	zero-page	DF	2	5	X X 0	Speichere X-Register (direct adressiert)
STSZ	Adresse	zero-page	9F	2	5	X X 0	Speichere Stapelzeiger (direct addr.)

Beispiel: Speichere das Indexregister nach MAX in Seite Null.

```
   Hexadezimal          Symbolisch
Adresse Inhalt   Name    Befehl    Operand

                         ORG       $0000
0000             MAX     RMB       2

                         ORG       $0180
0180     DF      START   STXZ      MAX
         00
```

Die gleiche Tätigkeit könnte auch durch den Befehl STX in absoluter Adressierung als 3-Byte-Befehl ausgeführt werden.
Der Funktionscode lautet dann FF , die Adresse besteht dann aus zwei Bytes.
Die zero-page Adressierung wird vom Hersteller direkte Adressierung genannt.

3.3.3 Das Speichern der 8080/8085-Register **8080**

Der 16 bit breite Adreßbus des 8080 bzw. 8085 erfordert zur Adressierung zwei Bytes, die hinter dem Funktionscode abgelegt werden (3-Byte-Befehl). Dabei stehen die wertniedrigsten Bits im zweiten Byte und die werthöchsten im dritten. Diese direkte Speicheradressierung - hier absolut genannt - läßt sich nur auf den Akkumulator (8 bit) und das Registerpaar H,L (16 bit) anwenden. Zum Speichern der anderen Register wird die Registerpaar-Adressierung verwendet; dabei steht die Adresse der Datenspeicherstelle nicht im Befehl, sondern in einem Registerpaar. Das L-Register gelangt in die adressierte Speicherstelle, das H-Register in die folgende.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C		
STA	Adresse	absolut	32	3	13			Speichere Akkumulator direkt
SHLD	Adresse	absolut	22	3	16			Speichere Registerpaar H,L direkt

Die Buchstaben STA bedeuten STore Accumulator = Speichere Akkumulator.
SHLD bedeutet Store H,L-Register Direct = Speichere H,L-Register direkt.

Beispiel: Speichere den Inhalt des Akkumulators in die Speicherstelle MAUSI

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	2000H
2000		MAUSI	DS	1
			ORG	2880H
2880	32	START	STA	MAUSI
	00			
	20			

Die anderen Register können über Register-Register-Operationen oder besser über die Registerpaar-Adressierung gespeichert werden. Eine zero-page Adressierung gibt es nicht.

3.4 Der Datentransport zwischen Speicherstellen

Daten können nicht direkt zwischen Daten-Speicherstellen oder Ein/Ausgabe-Registern transportiert werden. Es ist immer ein Register des Mikroprozessors als Zwischenspeicher zu verwenden.

Bild 75 zeigt den Transport eines Bytes von der Speicherstelle 0000 zur Speicherstelle 1111 mit Hilfe der Befehle "LADE Register" und "SPEICHERE Register". Mit einem 16-Bit-Register können zwei Bytes transportiert werden.

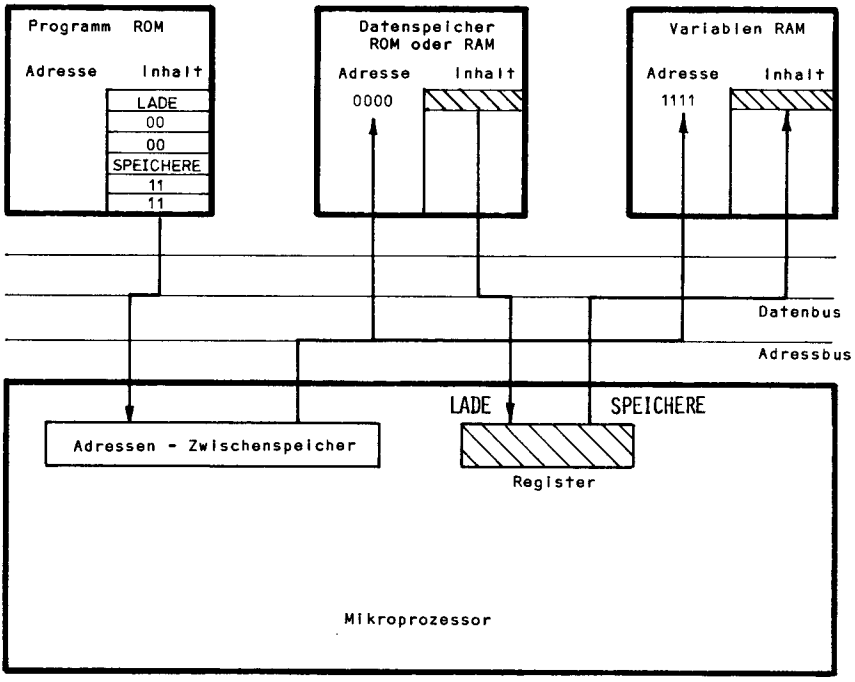


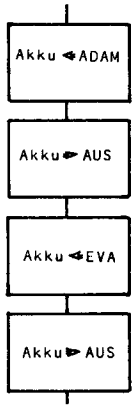
Bild 75: Datentransport zwischen zwei Speicherstellen

Nach Ausführung der Befehle haben die beiden Speicherstellen und das Register denselben Inhalt. Zur Übertragung größerer Speicherbereiche sind Schleifen mit Adreßrechnung vorteilhafter als mehrere Einzelbefehle.

3.5 Die symbolische Darstellung der Transportbefehle und Transportvorgänge

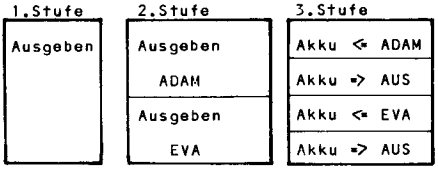
Der Programmablaufplan stellt jeden Befehl durch ein genormtes Symbol dar. Das Rechteck beschreibt das Laden und Speichern von Registern. Die Beschriftung ist nicht genormt, sie soll möglichst kurz sein.

Das folgende Beispiel zeigt eine Befehlsfolge, die die Speicherstellen ADAM und EVA in das Ausgaberegister AUS bringt.



Da sich alle Befehle auf ein Register beziehen (Laden und Speichern), steht links das Register und rechts die Speicherstelle. Das Zeichen <= bzw. => gibt die Transportrichtung an. Für die Darstellung der Ein/Ausgabe gibt es als Symbol das Parallelogramm, das im obigen Beispiel für die vier Befehle zu setzen wäre.

Der Programmblockplan beschreibt die Struktur eines Programms oder Programmteils und läßt sich nach Bedarf schrittweise verfeinern. Das folgende Beispiel zeigt, wie man den Programmblock "Ausgeben" in drei Schritten bis zum Programmablaufplan verfeinern kann.



Die Beispiele zeigen eine Folge von Befehlen bzw. Programmteilen.

Übungen zum Abschnitt 3.1 bis 3.5

Für die folgenden Aufgaben erstelle man einen Programmablaufplan, ein symbolisches Assemblerprogramm und ein hexadezimales Programm mit den in Abschnitt 2 vorgeschlagenen Adressen der Speicher und Ein/Ausgabe-Register.

1.Aufgabe

Man rette den Inhalt aller Register des Mikroprozessors in den Variablenspeicher.

2.Aufgabe

Man lösche alle Register des Mikroprozessors mit Ausnahme des Stapelzeigers mit binären Nullen.

3.Aufgabe

Man lade alle Register mit den in den Variablenspeicher geretteten Daten.

Für den Test der Übungsaufgaben ist es erforderlich, das Programm sinnvoll abzuschließen. Dazu kann man

a. wie im einführenden Beispiel das Programm in einer Schleife laufen lassen und mit Hilfe des Betriebssystems abbrechen oder
b. durch einen Sprungbefehl in das Betriebssystem zurückspringen.

3.6 Einführung in die Programmierung von Parallelschnittstellen

Parallelschnittstellen verbinden den parallel arbeitenden Datenbus mit äußeren ebenfalls parallel arbeitenden Schaltungen, die entweder Daten aus dem Mikrocomputer aufnehmen (z.B. Anzeigeeinheiten) oder an ihn abgeben (z.B. Eingabetasten). Die Schnittstellen nehmen zusätzlich Steuersignale auf und geben sie an die Interrupt-Leitungen des Mikrocomputers weiter. Läuft ein auszugebendes Datenbyte auf dem Datenbus, so kann es in den meisten Fällen nicht sofort von dem empfangenden Gerät aufgenommen werden. Ebenso ist es nicht möglich, von außen kommende Daten sofort über den Datenbus in den Prozessor zu transportieren. Aus diesem Grunde enthält die Schnittstelle Speicher in Form von Datenregistern. Um möglichst viele Anwendungsfälle mit einem Baustein erfassen zu können, sind die Funktionen der Schnittstelle programmierbar. Erst im Programm legt der Benutzer durch Befehle fest, welche Leitungen als Eingang oder als Ausgang geschaltet werden sollen und wie die Steuersignale wirken.

Bild 76 zeigt allgemein den Aufbau einer Parallelschnittstelle.

Die Schnittstelle enthält mehrere Register, die über Adressen ausgewählt werden.

Steuer- oder Kontrollregister nehmen vom Programm gesendete Steuerbytes auf, die die Funktion der Schnittstelle festlegen.
Richtungsregister bestimmen die Richtung (Eingabe oder Ausgabe) der Anschlußleitungen.
Datenregister speichern die Ein/Ausgabe-Daten.
Signalregister dienen als Zwischenspeicher für Steuersignale.
Die Adressen der Register werden beim Aufbau der Schaltung durch die Hardware festgelegt.

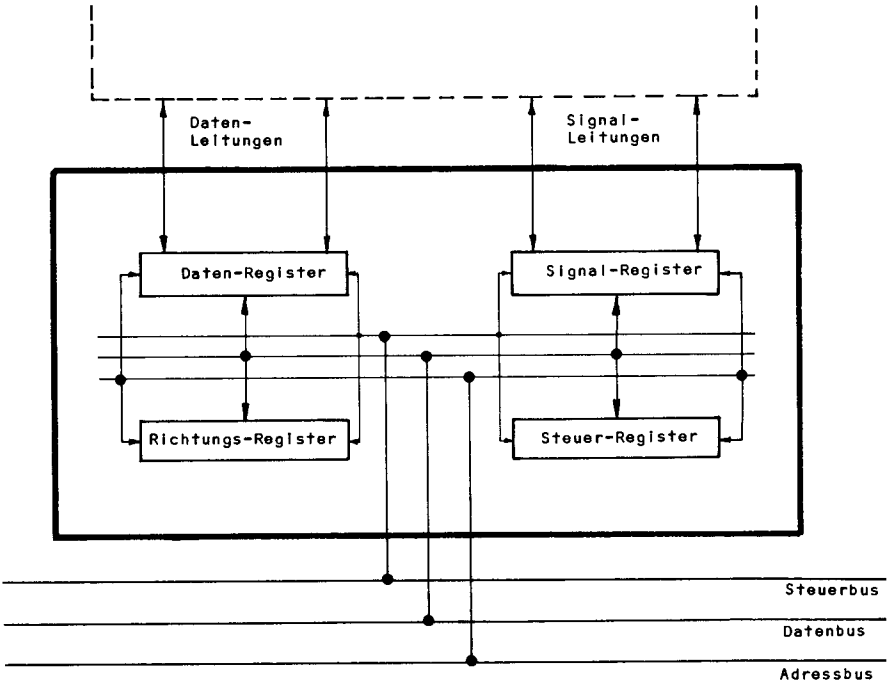


Bild 76: Aufbau einer programmierbaren Parallelschnittstelle

Die folgenden Beispiele programmieren eine Parallelschnittstelle des jeweiligen Mikrocomputers zum Anschluß an die Schaltung nach Bild 48. Dabei werden acht Leitungen als Eingang geschaltet, die ein Byte von Kippschaltern übernehmen. Über acht Ausgangsleitungen kann ein Byte über Leuchtdioden angezeigt werden.

3.6.1 Programmierung des Bausteins 6530(KIM-1) KIM

Bild 77 beschreibt den vereinfachten Aufbau der Parallelschnittstelle des Bausteins 6530 mit den Adressen der Register, so wie sie vom Hersteller des KIM-1 festgelegt wurden. Leitung B6 ist nicht angeschlossen, es stehen auf der B-Seite nur sieben Leitungen zur Verfügung.

Ein RESET löscht alle Register und schaltet damit alle Leitungen als Eingang. In diesem Zustand können keine Daten nach außen abgegeben werden. Die Leitungen liegen jedoch in diesem Zustand auf hohem Potential (ca. 4,5 V) und betätigen damit alle Geräte, die durch ein 1-Signal betrieben werden.
In der Anzeigeschaltung nach Bild 48 leuchten alle Anzeigen nach dem RESET auf.
Die Parallelschnittstelle des 6530 enthält zwei 8-Bit-Datenregister (A-Seite und B-Seite) mit den entsprechenden Datenrichtungsregistern. Diese vier Register können wie RAM-Speicherstellen unter den in Bild 77 angegebenen Adressen mit Lade- und Speicherbefehlen gelesen und beschrieben werden.

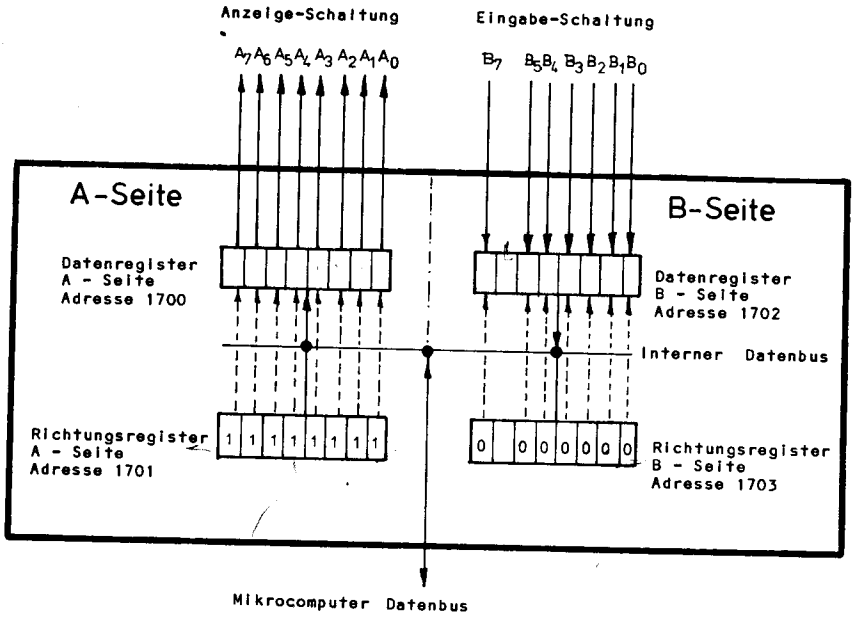


Bild 77: Parallelschnittstelle des Bausteins 6530

Der Inhalt des Datenrichtungsregisters bestimmt, welche Leitung des entsprechenden Datenregisters Eingang und welche Ausgang ist. Eine 0 in einem Bit des Datenrichtungsregisters schaltet die entsprechende Leitung des Datenregisters als Eingang, eine 1 schaltet die Leitung als Ausgang. Ist z.B. Bit 5 des Datenrichtungsregisters A-Seite eine 1, so ist Leitung A5 ein Ausgang.

Das folgende Beispiel programmiert entsprechend Bild 77 alle Leitungen der A-Seite als Ausgang und alle Leitungen der B-Seite als Eingang.

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
		RAU	=$1701	
		REI	=$1703	
			*=$0300	
0300	A9	START	LDA	#$00
	00			
0302	8D		STA	REI
	03			
	17			
0305	A9		LDA	#$FF
	FF			
0307	8D		STA	RAU
	01			
	17			

Zunächst werden die symbolischen Adressen RAU (Richtung AUsgabe) = 1701 und REI (Richtung EIngabe) = 1703 festgelegt. Die nächsten Befehle setzen das Richtungsregister der B-Seite auf 0 : die B-Seite ist Eingang.

Anschließend wird das Richtungsregister der A-Seite auf 1 gesetzt : die A-Seite ist Ausgang.
Nach dieser Festlegung können die Datenregister wie RAM-Speicherstellen gelesen und beschrieben werden.
Die Bits eines Registers können auch einzeln oder verschiedenartig verwendet werden.

3.6.2 Die Programmierung der Parallelschnittstelle 6820 (Bausatz 6800 D2 von Motorola) 6800

Bild 78 zeigt den vereinfachten Aufbau mit den Adressen der Register, so wie sie vom Hersteller des Mikrocomputers festgelegt wurden. Der Baustein heißt allgemein PIA (Peripheral Interface Adapter = Ein/Ausgabe-Anpassungs-Baustein).

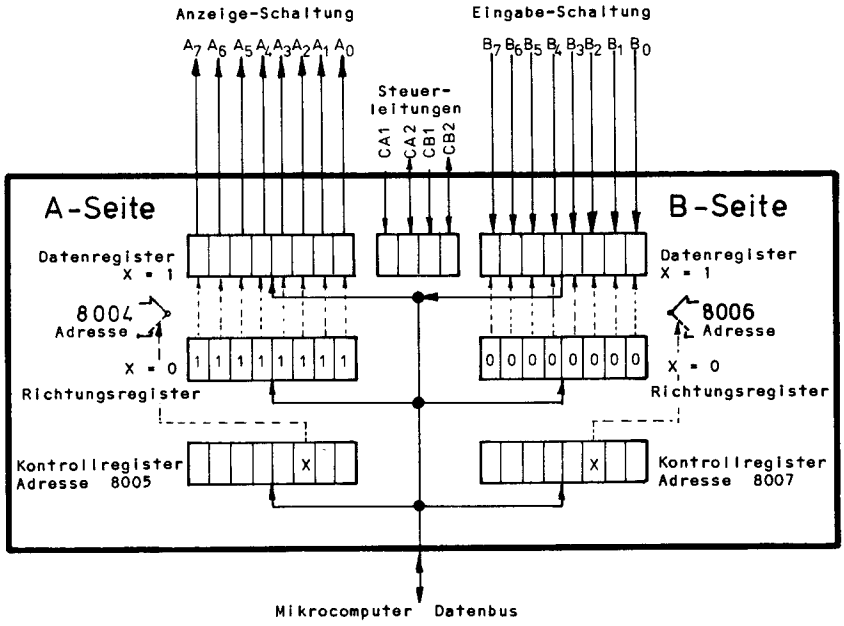


Bild 78: Parallelschnittstelle 6820

Ein RESET löscht alle Register und schaltet damit alle Leitungen als Eingang. In diesem Zustand können keine Daten nach außen abgegeben werden.
Die Leitungen der A-Seite liegen nach einem RESET auf hohem Potential (ca. 4,5 V). In der Schaltung Bild 48 sind alle Leuchtdioden hell. Die Leitungen der B-Seite sind nach dem RESET potentialfrei.
Jede Seite besteht aus drei Registern, einem Kontrollregister, einem Datenrichtungsregister und einem Datenregister.
Datenregister und Datenrichtungsregister haben eine gemeinsame Adresse, das Bit 2 des Kontrollregisters entscheidet, welches der beiden Register angesprochen wird.

Ist das Bit 2 des Kontrollregisters 0, so wird beim Aussenden der gemeinsamen Adresse das Datenrichtungsregister angesprochen; ist es 1, so wird das Datenregister adressiert.
Der Inhalt des Datenrichtungsregisters bestimmt, welche Leitung des entsprechenden Datenregisters Eingang und welche Ausgang ist. Eine 0 in einem Bit des Datenrichtungsregisters schaltet die entsprechende Leitung des Datenregisters als Eingang, eine 1 schaltet die Leitung als Ausgang. Ist z.B. Bit 5 des Datenrichtungsregisters A-Seite 1, so ist die Leitung A5 ein Ausgang.

Alle sechs Register können wie RAM-Speicherstellen unter den in Bild 78 angegebenen Adressen mit Lade- und Speicher-Befehlen gelesen und beschrieben werden.

Die Programmierung der Funktionen der Schnittstelle geschieht in folgenden Schritten:

a. Bit 2 des Kontrollregisters 0 setzen; damit wird das Richtungsregister adressiert.
b. Steuerbyte in das Richtungsregister bringen.
c. Bit 2 des Kontrollregisters 1 setzen; damit wird das Datenregister adressiert.

Das folgende Beispiel programmiert die A-Seite als Ausgang und die B-Seite als Eingang.

Hexadezimal		Symbolisch			
Adresse	Inhalt	Name	Befehl	Operand	BEMERKUNGEN
		AUS	EQU	$8004	
		KONA	EQU	$8005	
		EIN	EQU	$8006	
		KONB	EQU	$8007	
			ORG	$0180	
0180	7F	START	CLR	KONA	6 ZYKLEN
	80				
	05				
0183	86		LDA A	#$FF	2 ZYKLEN
	FF				
0185	B7		STA A	AUS	5 ZYKLEN
	80				
	04				
0188	C6		LDA B	#$04	2 ZYKLEN
	04				
018A	F7		STA B	KONA	5 ZYKLEN
	80				
	05				
018D	7F		CLR	KONB	6 ZYKLEN
	80				
	07				
0190	7F		CLR	EIN	6 ZYKLEN
	80				
	06				
0193	F7		STA B	KONB	5 ZYKLEN
	80				
	07				

Zuerst werden die symbolischen und hexadezimalen Adressen der Register festgelegt.
Anschließend wird das Kontrollregister der A-Seite und damit auch das Bit 2 gelöscht, das Richtungsregister auf 11111111 gesetzt und Bit 2 des Kontrollregisters auf 1 gebracht, so daß alle folgenden Speicherbefehle das Datenregister ansprechen.
Dann wird das Kontrollregister der B-Seite und damit auch das Bit 2 gelöscht, das Richtungsregister durch Löschen auf 00000000 gebracht und anschließend Bit 2 des Kontrollregisters auf 1 gesetzt, damit die folgenden Ladebefehle das Datenregister ansprechen.

Das Programmstück besteht aus 22 Bytes und benötigt 37 Zyklen.

Wird ein Programm nach einem RESET gestartet, das alle Register löscht, so kann es unter Benutzung des COM-Befehls = komplementiere wie folgt vereinfacht werden.

Hexadezimal		Symbolisch			
Adresse	Inhalt	Name	Befehl	Operand	BEMERKUNGEN
		AUS	EQU	$8004	
		KONA	EQU	$8005	
		EIN	EQU	$8006	
		KONB	EQU	$8007	
			ORG	$0180	
0180	73	START	COM	AUS	6 ZYKLEN
	80				
	04				
0183	86		LDA A	#$04	2 ZYKLEN
	04				
0185	B7		STA A	KONA	5 ZYKLEN
	80				
	05				
0188	B7		STA A	KONB	5 ZYKLEN
	80				
	07				

Dieses vereinfachte Programmstück besteht nur noch aus 11 Bytes und benötigt 18 Zyklen. Würde die Schnittstelle in Seite Null liegen, so wäre über die zero-page Adressierung eine weitere Vereinfachung möglich.
Die Bits eines Registers können auch einzeln und verschiedenartig verwendet werden.
Die Bedeutung der restlichen Bits des Kontrollregisters und die Programmierung der Steuerleitungen werden später erklärt.

3.6.3 Die Programmierung der Parallelschnittstelle 8255 (Bausatz SDK 8080 von Intel) **8080**

Bild 79 zeigt den vereinfachten Aufbau mit den Adressen der Register, so wie sie vom Hersteller des Entwicklungs-Bausatzes SDK 8080 festgelegt wurden.

Ein RESET bringt alle 24 Leitungen in den Eingabezustand. Der Baustein enthält ein Steuerregister und drei Ein/Ausgabe-Register mit je acht Leitungen. Das Steuerregister ist programmierbar und legt damit die Funktion der drei Kanäle fest.
Es gibt drei Betriebsarten:

Betriebsart 0 : einfache Ein/Ausgabe ohne Steuersignale
Betriebsart 1 : getastete Ein/Ausgabe mit Steuersignalen
Betriebsart 2 : getastete Zwei-Wege-Bus Ein/Ausgabe mit Steuersignalen.

Über ein Steuerbyte, das in das Steuerregister geschrieben wird, kann man durch das Programm festlegen, welche Gruppe von Leitungen ein Eingang und welche ein Ausgang ist. Die Programmierung einzelner Leitungen ist nicht möglich. Für die vier Leitungsgruppen gibt es 16 verschiedene Bitkombinationen.
Bild 80 zeigt die entsprechenden Steuerbytes.

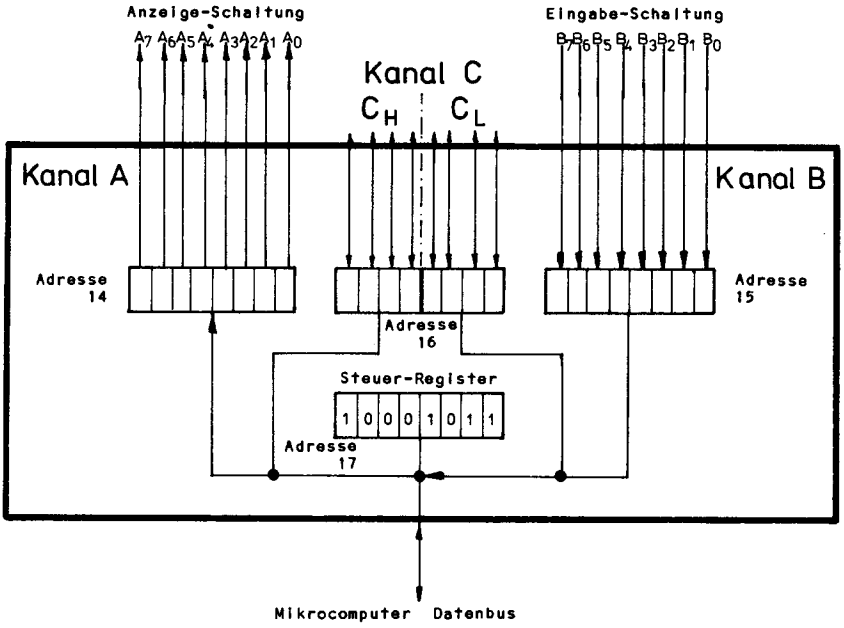


Bild 79: Parallelschnittstelle 8255

Kanal A	Kanal B	Kanal C_H	Kanal C_L	Steuerwort hexadezimal
AUS	AUS	AUS	AUS	80
AUS	AUS	AUS	EIN	81
AUS	AUS	EIN	AUS	88
AUS	AUS	EIN	EIN	89
AUS	EIN	AUS	AUS	82
AUS	EIN	AUS	EIN	83
AUS	EIN	EIN	AUS	8A
AUS	EIN	EIN	EIN	8B
EIN	AUS	AUS	AUS	90
EIN	AUS	AUS	EIN	91
EIN	AUS	EIN	AUS	98
EIN	AUS	EIN	EIN	99
EIN	EIN	AUS	AUS	92
EIN	EIN	AUS	EIN	93
EIN	EIN	EIN	AUS	9A
EIN	EIN	EIN	EIN	9B

Bild 80: Steuerbytes der Betriebsart 0

Beispiel: Kanal A soll als Ausgang, Kanal B und beide Teile des Kanals C sollen als Eingang dienen.

Das Steuerbyte lautet hexadezimal 8B, binär 1000 1011.
Die Leitungen des Bildes 79 wurden entsprechend diesem Steuerbyte beschriftet.

Zum Lesen und Schreiben der vier Register gibt es grundsätzlich zwei Möglichkeiten, die vom hardwaremäßigen Aufbau des betreffenden Mikrocomputers abhängen.

a. Die Ein/Ausgabe-Bausteine werden wie die übrigen Speicher (ROM und RAM) über einen Steuerbaustein an den Mikroprozessor angeschlossen und vom Programm wie Speicheradressen behandelt. Damit lassen sich alle den Speicher adressierenden Befehle auch auf die Register der Parallelschnittstelle anwenden, so z.B. die bisher behandelten Befehle

LDA	Lade Akkumulator
STA	Speichere Akkumulator
LHLD	Lade Registerpaar H,L
SHLD	Speichere Registerpaar H,L

Dieses Verfahren ist beim SDK 8080 nicht vorgesehen.

b. Die Ein/Ausgabe-Bausteine werden von einem Steuerbaustein über besondere Ein/Ausgabe-Steuerleitungen an den Mikroprozessor angeschlossen. Die Adressen der Register werden in einem Byte untergebracht und sind Hexadezimalzahlen zwischen 00 und FF . Die beiden folgenden Befehle verbinden den Akkumulator mit den Ein/Ausgabe-Kanälen (Registern).

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
IN	Kanal-Nr	E/A-Kanal	DB	2	10			Lade Akkumulator mit Eingabe-Kanal
OUT	Kanal-Nr	E/A-Kanal	D3	2	10			Speichere Akkumulator nach Ausgabe-Kanal

IN bedeutet INput = Eingabe, OUT bedeutet OUTput = Ausgabe.

Beispiel: Das Steuerbyte 8B ist in das Steuerregister Kanal-Nr. 17 zu bringen.

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	1380H
1380	3E	START	MVI	A,8BH
	8B			
1382	D3		OUT	17H
	17			

3.6.4 Die Programmierung der Parallelschnittstelle 8155 (Entwicklungs-Bausatz SDK 8085)

Der Baustein 8155 enthält neben einem 256 X 8 bit RAM und einem Zeitgeber (Timer) drei Ein/Ausgabe-Register. Bild 81 zeigt den vereinfachten Aufbau mit den Adressen der Register, so wie sie vom Hersteller des Bausatzes festgelegt wurden.

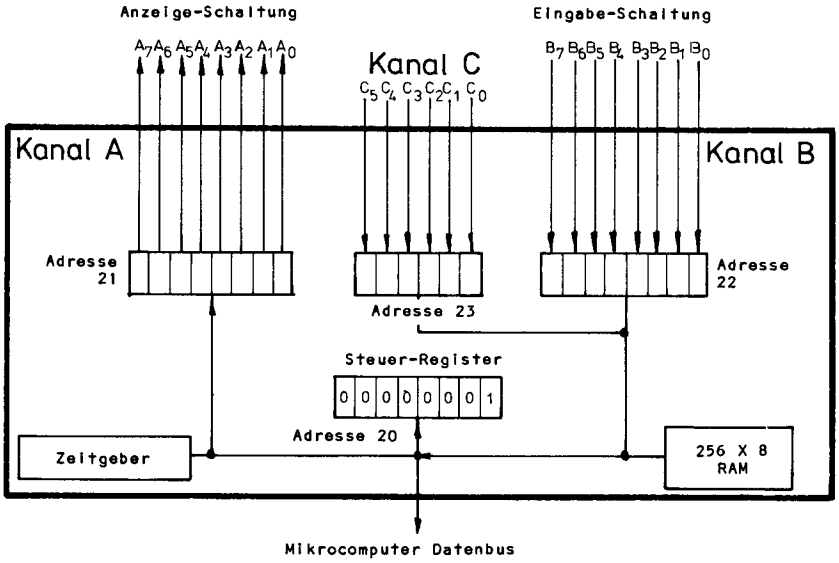


Bild 81: Parallelschnittstelle 8155

Ein RESET bringt alle Leitungen in den Eingabezustand. Der Baustein enthält zwei 8-Bit-Ein/Ausgabe-Register (Kanal A und B) und ein 6-Bit-Register, das wahlweise zur Ein/Ausgabe oder für Steuersignale verwendet werden kann. Ein Steuerregister legt die Funktionen der drei Kanäle fest. Dieser Abschnitt behandelt nur den einfachen Fall, daß der Kanal C für die Ein/Ausgabe verwendet wird. Bild 82 zeigt den Aufbau des Steuerbytes für die verschiedenen Kombinationen.

Kanal A	Kanal B	Kanal C	Steuerwort hexadezimal
AUS	AUS	AUS	0F
AUS	AUS	EIN	03
AUS	EIN	AUS	0D
AUS	EIN	EIN	01
EIN	AUS	AUS	0E
EIN	AUS	EIN	02
EIN	EIN	AUS	0C
EIN	EIN	EIN	00

Bild 82: Steuerbytes für einfache Ein/Ausgabe

Die vier Register werden unter den in Bild 81 eingezeichneten Kanal-Nummern mit den Befehlen IN und OUT angesprochen.

Beispiel: Kanal A soll als Ausgang, die Kanäle B und C als Eingang programmiert werden.
Das Steuerbyte lautet hexadezimal 01 , binär 0000 0001 .
Das folgende Programmstück lädt das Steuerbyte in das Steuerregister. Bild 81 wurde entsprechend beschriftet.

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,01H
	01			
2882	D3		OUT	20H
	20			

3.6.5 Die Programmierung der Parallelschnittstelle 8355 (Entwicklungs-Bausatz SDK 8085)

Der Baustein enthält neben einem 2K X 8 ROM mit dem Betriebsprogramm zwei Ein/Ausgabe-Register von je acht bit Länge. Bild 83 zeigt den vereinfachten Aufbau mit den Adressen der Register, so wie sie vom Hersteller des Bausatzes festgelegt wurden.

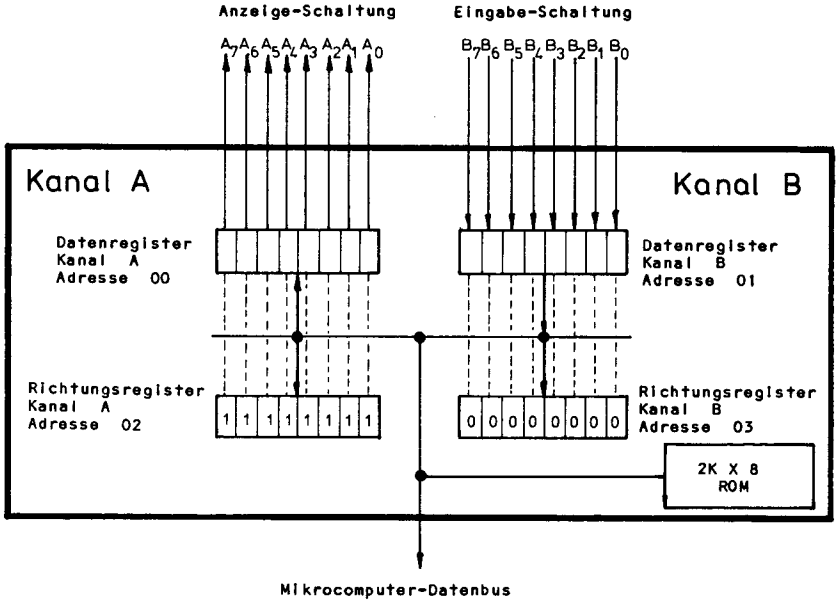


Bild 83: Parallelschnittstelle 8355

Ein RESET löscht alle Register und schaltet damit alle Leitungen als Eingang. Die vier Register werden unter den in Bild 83 gezeigten Kanal-Nummern mit den Befehlen IN und OUT angesprochen.

Der Inhalt des Datenrichtungsregisters bestimmt, welche Leitung des entsprechenden Datenregisters Eingang und welche Ausgang ist. Eine 0 in einem Bit des Richtungsregisters schaltet die entsprechende Leitung des Datenregisters als Eingang, eine 1 schaltet die entsprechende Leitung als Ausgang. Ist z.B. Bit 5 des Datenrichtungsregisters der A-Seite eine 1, so ist die Leitung A5 ein Ausgang.

Beispiel: Das folgende Programmstück programmiert die A-Seite als Ausgang und die B-Seite als Eingang.

Hexadezimal		Symbolisch		
Adresse	Inhalt	Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	D3		OUT	02H
	02			
2884	3E		MVI	A,00H
	00			
2886	D3		OUT	03H
	03			

Übungen zum Abschnitt 3.6. A

1.Aufgabe
Man programmiere alle verfügbaren Leitungen der Parallelschnittstelle als Eingang und übertrage die anliegenden Daten in verschiedene Bytes des Variablenbereiches.

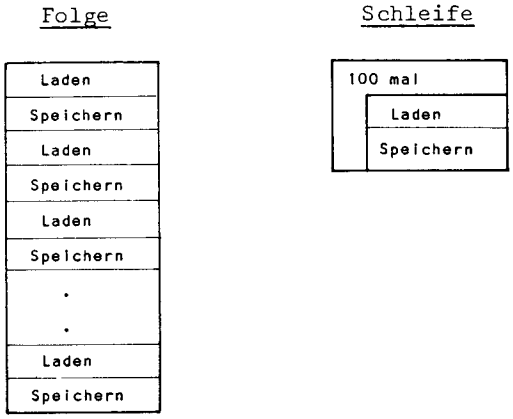
2.Aufgabe
Man programmiere alle verfügbaren Leitungen der Parallelschnittstelle als Ausgang und lege alle Leitungen auf hohes Potential (Logisch 1).

3.Aufgabe
Man programmiere acht Leitungen der Parallelschnittstelle als Eingang und acht Leitungen als Ausgang und schreibe ein Programm, das die Eingangsdaten zum Ausgang transportiert.

4. Die Sprungbefehle und ihre Anwendungen in Schleifen A

Die im vorigen Abschnitt behandelten Programme bestanden aus einer Folge von Befehlen; bei der Übertragung von z.B. 100 Bytes aus dem Datenspeicher in ein Ausgaberegister benötigt man in dieser Programmiertechnik 100 Lade- und 100 Speicherbefehle.

Dieser Abschnitt zeigt nun, wie man in einer Schleife ein Programmteil - bestehend z.B. aus einem Lade- und einem Speicherbefehl - beliebig oft ausführen kann.



Zur Übertragung von Speicherbereichen sind zusätzliche Adreßrechnungen erforderlich.
Der Abschnitt 5 beschreibt die Anwendung der Sprungbefehle in Programmverzweigungen.

4.1 Der Ablauf einer Befehlsfolge

Befehlsfolgen werden Befehl für Befehl nacheinander ausgeführt. Das Befehlszählregister des Mikroprozessors enthält immer die Adresse des nächsten aus dem Programmspeicher zu holenden Bytes. Bild 84 zeigt als Beispiel eine Folge von drei Befehlen. Um die verschiedenen Arten der Befehlsausführung zu zeigen, besteht die Folge aus einem 1-Byte-Befehl, einem 2-Byte-Befehl und einem 3-Byte-Befehl.

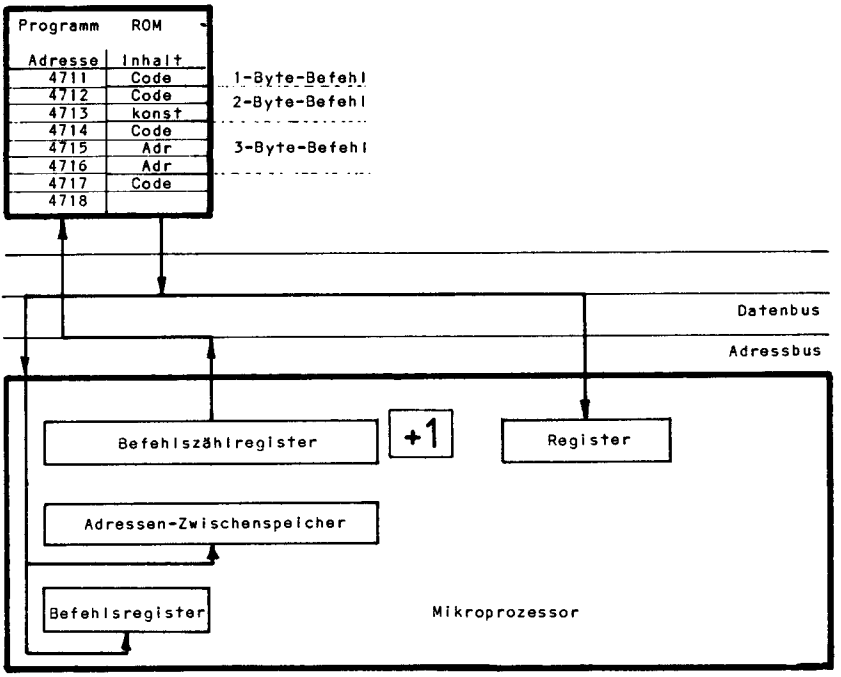


Bild 84: Anordnung einer Befehlsfolge im Speicher

Das Befehlszählregister enthalte als Ausgangsadresse die Zahl 4711, die laufend um 1 erhöht wird. Bild 85 zeigt schematisch, wie die verschiedenen Teile des Befehls in den Mikroprozessor transportiert werden. Der Code gelangt in das Befehlsregister, Konstanten in die Datenregister und Adressen in einen dem Programmierer nicht zugänglichen Zwischenspeicher.

Takt	Befehlszählerstand	Ausführung
1	4711	Adresse 4711 zum Programmspeicher
2	4712	Code zum Befehlsregister
3	4712	Adresse 4712 zum Programmspeicher
4	4713	Code zum Befehlsregister
5	4713	Adresse 4713 zum Programmspeicher
6	4714	Konstante zum Register
7	4714	Adresse 4714 zum Programmspeicher
8	4715	Code zum Befehlsregister
9	4715	Adresse 4715 zum Programmspeicher
10	4716	1. Teil der Datenadresse holen
11	4716	Adresse 4716 zum Programmspeicher
12	4717	2. Teil der Datenadresse holen
13	4717	gesamte Datenadresse aussenden
14	4717	Adresse 4717 zum Programmspeicher
15	4718	Code zum Befehlsregister

Bild 85: Die Ausführung einer Befehlsfolge

4.2 Der unbedingte Sprung

Bei einem unbedingten Sprung wird das Befehlszählregister mit einer neuen Befehlsadresse geladen. Der Sprungbefehl enthält neben dem Funktionscode die Adresse des neuen Befehls. Bild 86 zeigt als Beispiel einen Sprung zum Befehl mit der Adresse 2222.

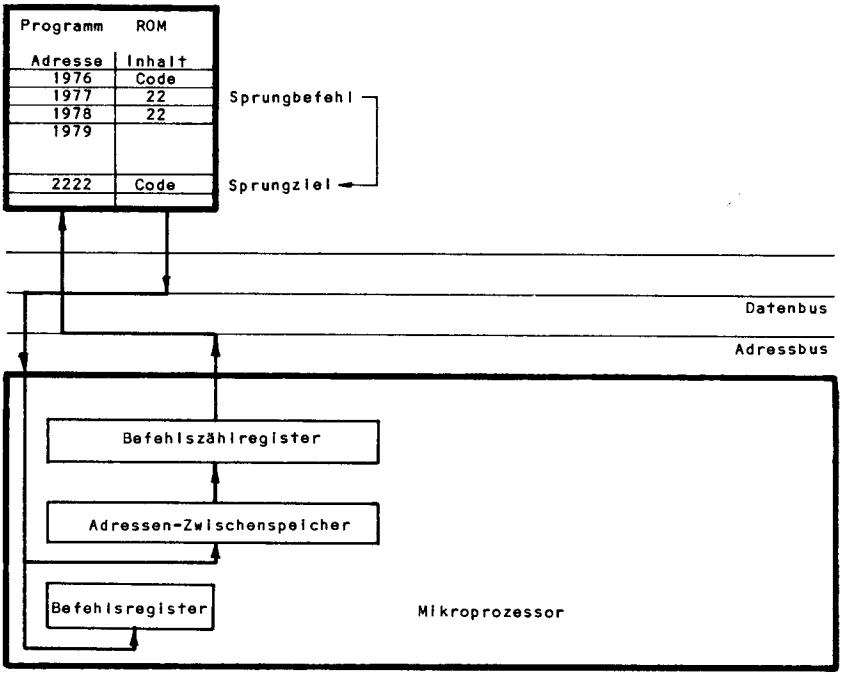


Bild 86: Sprungbefehl und Sprungziel im Programmspeicher

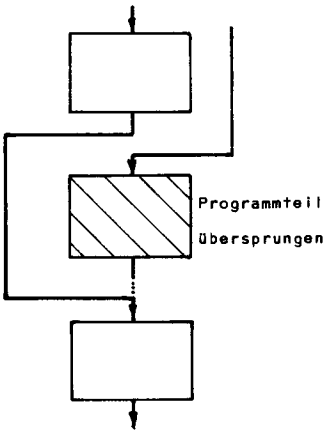
Das Befehlszählregister enthalte als Ausgangsadresse die Zahl 1976, die durch die Adresse 2222 ersetzt wird. Bild 87 zeigt schematisch den Ablauf. Der Code gelangt in das Befehlsregister, die neue Befehlsadresse zunächst in einen Zwischenspeicher und von dort in das Befehlszählregister.

Takt	Befehlszählerstand	Ausführung
1	1976	Adresse 1976 zum Programmspeicher
2	1977	Code zum Befehlsregister
3	1977	Adresse 1977 zum Programmspeicher
4	1978	1. Teil der Sprungadresse holen
5	1978	Adresse 1978 zum Programmspeicher
6	1979	2. Teil der Sprungadresse holen
7	1979	Sprungadresse nach Befehlszähler
8	2222	Adresse 2222 zum Programmspeicher
9	2223	Code zum Befehlsregister

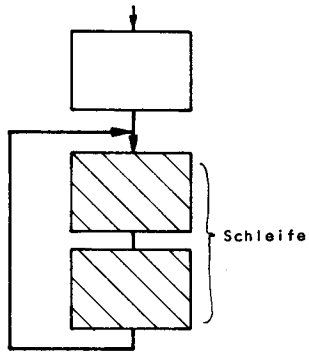
Bild 87: Ausführung eines Sprungbefehls

Im Programmablaufplan wird der unbedingte Sprung durch eine Verbindungslinie mit Pfeil zum Sprungziel dargestellt. Man unterscheidet Vorwärtssprünge und Rückwärtssprünge.

Vorwärtssprung



Rückwärtssprung



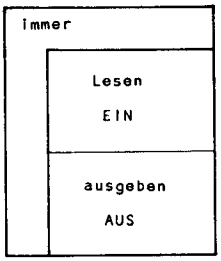
Wird bei einem Rückwärtssprung ein Programmteil mehrmals durchlaufen, so spricht man von einer Schleife.
Im Programmblockplan kann man die Schleife durch folgendes Symbol darstellen:

Programmblockplan einer Schleife

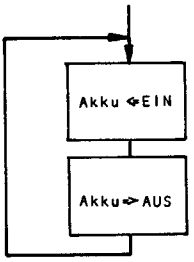
Mit einem unbedingten Sprung programmierte Schleifen können nur hardwaremäßig durch einen Interrupt oder ein RESET abgebrochen werden. Bei einem RESET wird das Programm neu gestartet, nach einem Interrupt läuft ein Unterbrechungsprogramm ab. Arbeitet man mit einem Entwicklungssystem, so kann man durch Betätigen einer entsprechenden Taste in das Betriebssystem zurückkehren.

Das folgende Beispiel zeigt den Programmblockplan und den Programmablaufplan einer Schleife, die laufend Daten vom Eingaberegister EIN liest und am Ausgaberegister AUS wieder ausgibt.

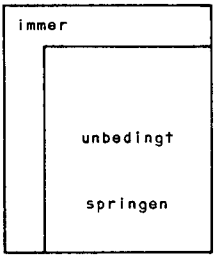
Programmblockplan

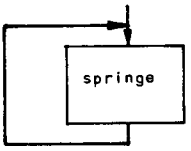


Programmablaufplan



Die kürzeste Schleife besteht nur aus einem unbedingten Sprung zu diesem Befehl selbst. Sie kann dazu benutzt werden, bei der hardwaremäßigen Untersuchung eines Mikrocomputers eine kurze immerwiederkehrende (periodische) Signalfolge auf dem Adreß- und Datenbus zu erzeugen, die sich leicht oszillografieren läßt.





4.2.1 Der unbedingte Sprungbefehl des KIM-1 KIM

Der unbedingte Sprungbefehl des KIM-1 verlangt die Angabe einer 2-Byte-Sprungadresse. Die wertniedrigsten Bits stehen im zweiten Byte des Befehls, die werthöchsten im dritten. Das Befehlszählregister wird mit dieser neuen Adresse überschrieben, der alte Inhalt geht verloren, der Mikroprozessor verarbeitet das unter der neuen Adresse abgelegte Byte als Funktionscode eines neuen Befehls.
In der Assemblerschreibweise steht hinter dem Befehl die symbolische Adresse des danach auszuführenden Befehls.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	
JMP	Adresse	absolut	4C	3	3		Führe immmer den adressierten Befehl aus

JMP bedeutet JuMP = springe.

Beispiel:
In einer Schleife werden fortlaufend Daten von einem Eingaberegister EIN gelesen und über ein Ausgaberegister AUS ausgegeben. Ein hier nicht behandeltes Programmstück programmiert die Parallelschnittstelle.

```
  Hexadezimal                Symbolisch
Adresse Inhalt   Name    Befehl    Operand

                 EIN     =$1702
                 AUS     =$1700

                         *=$0300
0300    AD       SUSI    LDA       EIN
        02
        17
0303    8D               STA       AUS
        00
        17
0306    4C               JMP       SUSI
        00
        03
                         .END
```

Das folgende Beispiel zeigt eine Warteschleife, die nur durch ein RESET oder Interrupt verlassen werden kann.

```
  Hexadezimal                Symbolisch
Adresse Inhalt   Name    Befehl    Operand

                         *=$0300
0300    4C       OTTO    JMP       OTTO
        00
        03
                         .END
```

4.2.2 Die unbedingten Sprungbefehle des 6800 **6800**

Der Mikroprozessor 6800 kennt zwei verschiedene Möglichkeiten, das Befehlszählregister mit einer neuen Adresse zu laden.

1. Den unbedingten Sprungbefehl mit einer 2-Byte-Sprungadresse.
2. Den unbedingten Verzweigungsbefehl mit einer 1-Byte-Sprungweite.

Der unbedingte Sprungbefehl JMP lädt die im zweiten und dritten Byte des Befehls enthaltene Adresse des Sprungziels in das Befehlszählregister. Dabei stehen die werthöchsten Bits der Adresse im zweiten Byte, die wertniedrigsten im dritten Byte des Befehls. Der alte Inhalt des Befehlszählregisters geht verloren, das unter der neuen Adresse abgespeicherte Byte wird vom Mikroprozessor als Funktionscode eines neuen Befehls verarbeitet.

In der Assemblerschreibweise steht hinter dem Befehl die symbolische Adresse des danach auszuführenden Befehls.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI NZVC	6800
JMP	Adresse	absolut	7E	3	3		Führe immer den adressierten Befehl aus

JMP bedeutet JuMP = springe.

Beispiel:
In einer Schleife werden fortlaufend Daten von einem Eingaberegister EIN gelesen und über ein Ausgaberegister AUS ausgegeben. Dabei wird der unbedingte Sprungbefehl benutzt. Ein hier nicht behandeltes Programmstück programmiert vorher die Parallelschnittstelle.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl    Operand

                AUS    EQU       $8004
                EIN    EQU       $8006

                       ORG       $0180
0180      B6    SUSI   LDA A     EIN
          80
          06
0183      B7           STA A     AUS
          80
          04
0186      7E           JMP       SUSI
          01
          80
                       END
```

Das folgende Beispiel zeigt eine Warteschleife mit dem unbedingten Sprungbefehl, die nur durch RESET oder einen Interrupt verlassen werden kann.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl    Operand

                       ORG       $0180
0180      7E    OTTO   JMP       OTTO
          01
          80
                       END
```

Der unbedingte Verzweigungsbefehl **BRA** ist ein 2-Byte-Befehl. Das erste Byte enthält den Funktionscode, das zweite Byte eine Dualzahl mit Vorzeichen - die Sprungweite -, die über ein Adreßrechenwerk zum Inhalt des Befehlszählers dazuaddiert wird. Die Summe aus altem Befehlszählerstand und Sprungweite gibt die Adresse des neuen Befehls. Ist die Sprungweite positiv, so erhöht sich der Befehlszähler, es entsteht ein Vorwärtssprung. Ist die Sprungweite negativ, so vermindert sich der Befehlszähler, es entsteht ein Rückwärtssprung. Diese Adressierungsart, bei der im Befehl eine Sprungweite steht, heißt relative Adressierung. Sie wird beim 6800 für alle bedingten Sprünge verwendet. Bild 88 zeigt das Schema der relativen Adressierung mit Vorwärtssprung.

Bei der Berechnung der relativen Adresse (Sprungweite) ist zu beachten, daß vor der Adreßrechnung der Befehlszähler bereits auf der Adresse des nächsten Befehls steht.

Beispiel:

```
4713   Befehlszählerstand
 +06   Sprungweite
4719   Adresse des Sprungziels
```

Bild 89 zeigt das Schema der relativen Adressierung mit einem Rückwärtssprung.

Bei der Berechnung der Sprungweite (relative Adresse) ist zu beachten, daß vor der Adreßrechnung der Befehlszähler bereits auf der Adresse des nächsten Befehls steht.

Beispiel:

```
1979   Befehlszählerstand
 -08   Sprungweite
1971   Adresse des Sprungziels
```

Negative Sprungweiten sind im Zweierkomplement darzustellen. Bei der hexadezimalen Programmierung muß der Programmierer die Sprungweite hexadezimal angeben. Am einfachsten ist das dezimale Abzählen.

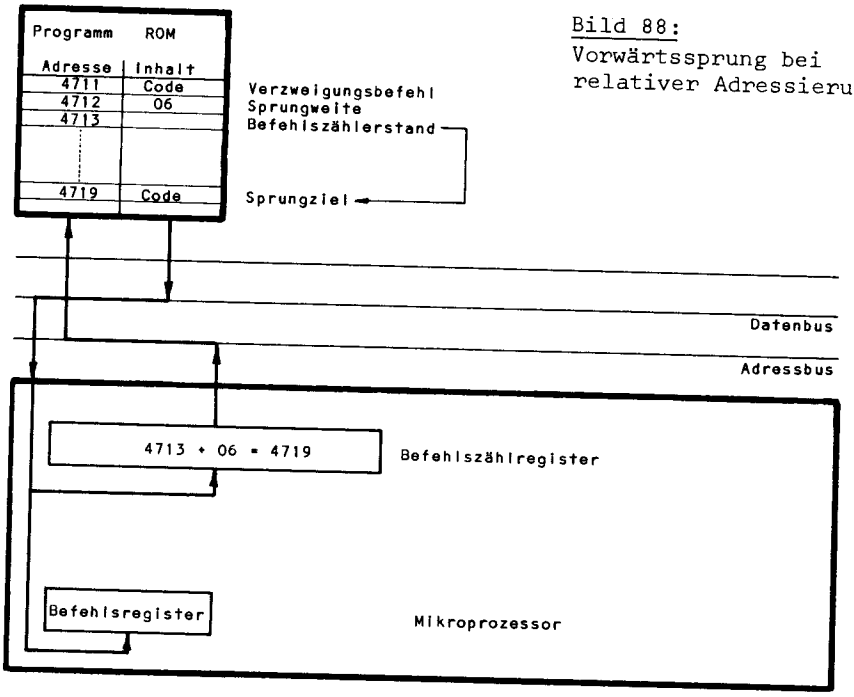


Bild 88:
Vorwärtssprung bei relativer Adressierung

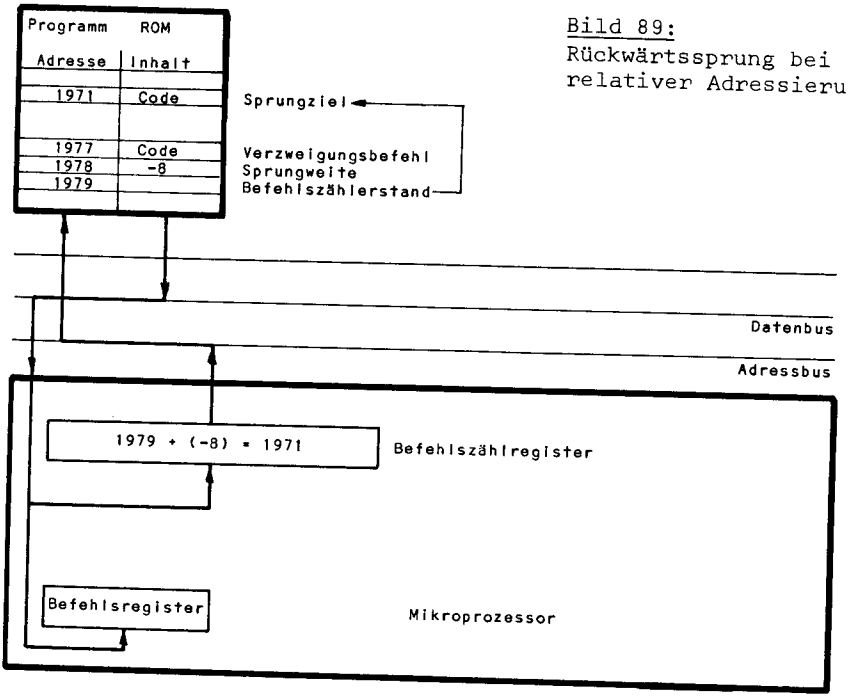


Bild 89:
Rückwärtssprung bei relativer Adressierung

Ausgehend von der Adresse des nächsten Befehls zählt man die Bytes bis zum Sprungziel und wandelt die Dezimalzahl - gegebenenfalls unter Zuhilfenahme der Tabelle Anlage 3 - in eine Hexadezimalzahl um.

Beispiel: Die Sprungweite -8_{10} gibt $F8_{16}$

Bei der symbolischen Adressierung enthält der Operandenteil eine symbolische Sprungadresse, der Assembler berechnet die Sprungweite und setzt sie in das zweite Byte des Befehls ein. Die Sprungweite ist auf 127 Bytes vorwärts und 128 Bytes rückwärts beschränkt, da nur sieben Bits und ein Vorzeichen-Bit zur Verfügung stehen. Verzweigungen über größere Entfernungen benötigen den JMP-Befehl.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI NZVC	6800
BRA	Adresse	relativ	20	2	4		Verzweige immer zum adressierten Befehl

BRA bedeutet BRAnch = Verzweige.

Beispiel:
In einer Schleife werden fortlaufend Daten von einem Eingaberegister EIN gelesen und über ein Ausgaberegister AUS ausgegeben. Dabei wird nun der unbedingte Verzweigungsbefehl benutzt. Ein hier nicht behandeltes Programmstück programmiert vorher die Parallelschnittstelle.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name    Befehl    Operand

                AUS     EQU       $8004
                EIN     EQU       $8006

                        ORG       $0180
0180      B6    SISSI   LDA A     EIN
          80
          06
0183      B7            STA A     AUS
          80
          04
0186      20            BRA       SISSI
          F8
0188                    END
```

Das folgende Beispiel zeigt eine Warteschleife mit dem unbedingten Verzweigungsbefehl, die nur durch RESET oder einen Interrupt verlassen werden kann.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name    Befehl    Operand

                        ORG       $0180
0180      20    OTTO    BRA       OTTO
          FE
0182                    END
```

4.2.3 Der unbedingte Sprungbefehl des 8080/8085 **8080**

Der unbedingte Sprungbefehl des 8080 bzw. 8085 verlangt die Angabe einer 2-Byte-Sprungadresse. Die wertniedrigsten Bits stehen im zweiten Byte des Befehls, die werthöchsten im dritten. Das Befehlszählregister wird mit dieser Adresse überschrieben,

der alte Inhalt geht verloren; der Mikroprozessor verarbeitet das unter dieser neuen Adresse abgelegte Byte als Funktionscode eines neuen Befehls.

In der Assemblerschreibweise steht hinter dem Befehl JMP die symbolische Adresse des danach auszuführenden Befehls.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H -P-C	
JMP	Adresse	absolut	C3	3	10		Führe immer den adressierten Befehl aus

JMP bedeutet JuMP = Springe.

Beispiel:
In einer Schleife werden fortlaufend Daten von einem Eingaberegister 01 gelesen und über ein Ausgaberegister 00 ausgegeben. Ein hier nicht behandeltes Programmstück programmiert vorher die Parallelschnittstelle.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2880H
2880    DB      SUSI   IN       01H
        01
2882    D3             OUT      00H
        00
2884    C3             JMP      SUSI
        80
        28
                       END
```

Das folgende Beispiel zeigt eine Warteschleife, die nur durch ein RESET oder einen Interrupt verlassen werden kann.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2880H
2880    C3      OTTO   JMP      OTTO
        80
        28
                       END
```

4.3 Der bedingte Sprung A

Der bedingte Sprung verzweigt bzw. springt nur dann zu einem neuen Befehl, wenn eine Sprungbedingung erfüllt ist; sonst wird der nächste Befehl ausgeführt. Die Sprungbedingungen beziehen sich immer auf ein bestimmtes Bit des Bedingungsregisters, das vom Steuerwerk untersucht und ausgewertet wird.
Bild 90a zeigt, wie bei einer erfüllten Sprungbedingung zu dem in der Adresse genannten Befehl gesprungen wird, Bild 90b zeigt, daß bei einer nicht erfüllten Sprungbedingung der nächste Befehl zur Ausführung kommt.

Im Programmablaufplan stellt man den bedingten Sprung durch einen Rhombus dar. Es ist das einzige Symbol mit einem Eingang und zwei Ausgängen. Man unterscheidet wieder den Vorwärts- und den Rückwärtssprung.

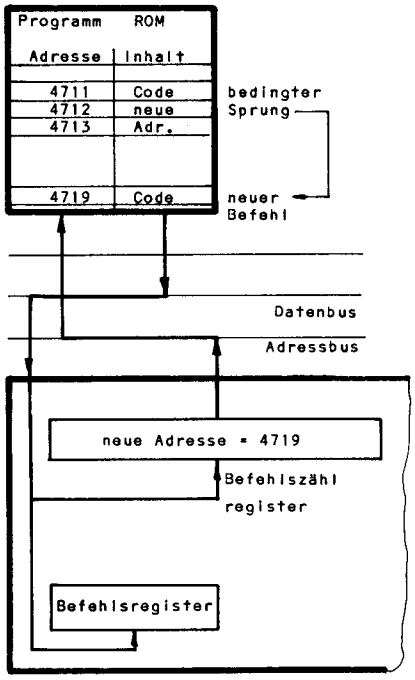


Bild 90a:
Sprungbefehl wird ausgeführt

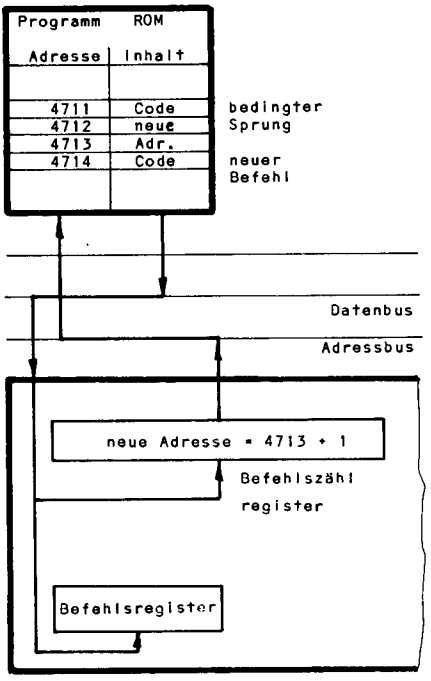
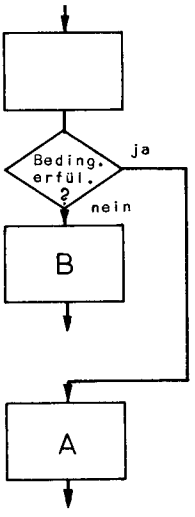
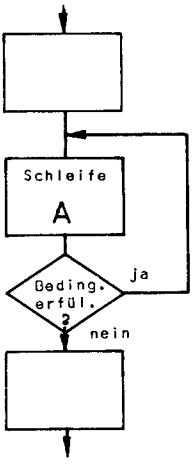


Bild 90b:
Sprungbefehl wird nicht ausgeführt

Vorwärtssprung

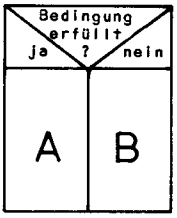


Rückwärtssprung

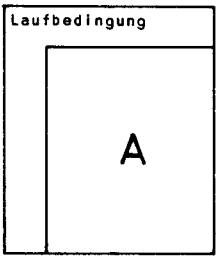


Der Vorwärtssprung ergibt eine Programmverzweigung, der Rückwärtssprung eine Schleife mit einer Laufbedingung. Sie werden im Programmblockplan durch folgende Symbole dargestellt.

Programmverzweigung



Programmschleife



4.3.1 Die bedingten Sprünge des KIM-1 KIM

Alle bedingten Sprünge sind Verzweigungsbefehle von zwei byte Länge. Das erste Byte enthält den Funktionscode, das zweite Byte eine Dualzahl mit Vorzeichen - die Sprungweite -, die über ein Adreßrechenwerk zum Inhalt des Befehlszählers dazuaddiert wird. Die Summe aus altem Befehlszählerstand und Sprungweite gibt die Adresse des neuen Befehls. Ist die Sprungweite positiv, so erhöht sich der Befehlszähler, es entsteht ein Vorwärtssprung. Ist die Sprungweite negativ, so vermindert sich der Befehlszähler, es entsteht ein Rückwärtssprung. Diese Adressierungsart, bei der im Befehl eine Sprungweite steht, heißt relative Adressierung.

Bild 91 zeigt das Schema der relativen Adressierung mit einem Vorwärtssprung, Bild 92 die relative Adressierung mit einem Rückwärtssprung.
Bei der Berechnung der relativen Adresse (Sprungweite) ist zu beachten, daß vor der Adreßrechnung der Befehlszähler bereits auf der Adresse des nächsten Befehls steht.

Beispiele:

```
4713   Befehlszählerstand
 +06   Sprungweite
4719   Adresse des Sprungziels

1979   Befehlszählerstand
 -08   Sprungweite
1971   Adresse des Sprungziels
```

Negative Sprungweiten sind im Zweierkomplement darzustellen. Bei der hexadezimalen Programmeingabe muß der Programmierer die Sprungweite hexadezimal berechnen. Am einfachsten ist das dezimale Abzählen.

Ausgehend von der Adresse des nächsten Befehls zählt man die Bytes bis zum Sprungziel und wandelt die Dezimalzahl - gegebenenfalls unter Zuhilfenahme der Tabelle Anhang 3 - in eine Hexadezimalzahl um.

Beispiel: Die Sprungweite -8_{10} gibt $F8_{16}$

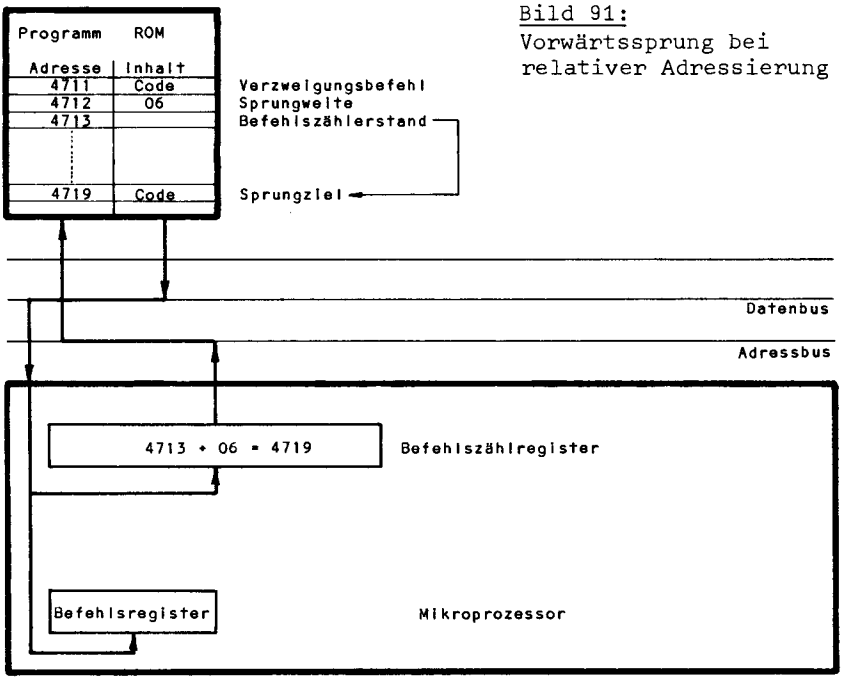


Bild 91:
Vorwärtssprung bei relativer Adressierung

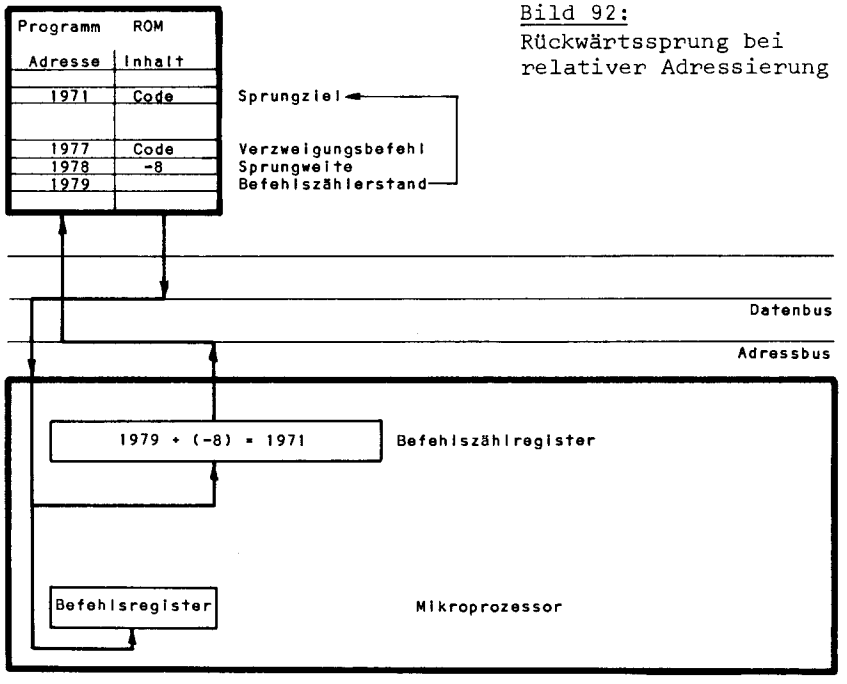


Bild 92:
Rückwärtssprung bei relativer Adressierung

Bei der symbolischen Programmierung enthält das Operandenfeld die symbolische Sprungadresse, der Assembler berechnet die Sprungweite und setzt sie in das zweite Byte des Befehls ein. Die Sprungweite ist auf 127 Bytes vorwärts und 128 Bytes rückwärts beschränkt, da im zweiten Byte des Befehls nur sieben Bits und ein Vorzeichenbit zur Verfügung stehen, um die Sprungweite aufzunehmen. Verzweigungen über größere Entfernungen müssen den unbedingten Sprung JMP zu Hilfe nehmen.
Die Sprungbedingungen beziehen sich auf einzelne Bits des Bedingungsregisters, die von den vorhergehenden Befehlen gesetzt werden.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	KIM
BCC	Adresse	relativ	90	2	2/3/4		Verzweige bei C=0 Ergebnis ohne Übertrag
BCS	Adresse	relativ	B0	2	2/3/4		Verzweige bei C=1 Ergebnis mit Übertrag
BEQ	Adresse	relativ	F0	2	2/3/4		Verzweige bei Z=1 Ergebnis gleich Null
BNE	Adresse	relativ	D0	2	2/3/4		Verzweige bei Z=0 Ergebnis ungleich Null
BMI	Adresse	relativ	30	2	2/3/4		Verzweige bei N=1 Ergebnis negativ
BPL	Adresse	relativ	10	2	2/3/4		Verzweige bei N=0 Ergebnis positiv
BVC	Adresse	relativ	50	2	2/3/4		Verzweige bei V=0 Ergebnis ohne Überlauf
BVS	Adresse	relativ	70	2	2/3/4		Verzweige bei V=1 Ergebnis mit Überlauf

Der Buchstabe B ist eine Abkürzung für Branch = Verzweige. Der Befehl BCC bedeutet Branch on Carry Clear = Verzweige, wenn das Carry-Bit gelöscht ist.

Für die Adreßrechnung wird ein 8-Bit-Addierer verwendet. Bei der Ausführung eines bedingten Verzweigungsbefehls gibt es drei Möglichkeiten:

a. Die Sprungbedingung ist nicht erfüllt, die im Befehl enthaltene Sprungweite wird nicht verwendet, da der alte Befehlszählerstand erhalten bleibt. Die Ausführung des Befehls dauert zwei Zyklen.
b. Die Sprungbedingung ist erfüllt, die im Befehl enthaltene Sprungweite wird zunächst zum wertniedrigsten Byte des Befehlszählers addiert. Erfolgt kein Übertrag auf das werthöhere Byte, so ist damit die Adreßrechnung beendet, der Befehl wird in drei Zyklen ausgeführt.
c. Die Sprungbedingung ist erfüllt, bei der Adreßrechnung entsteht ein Übertrag auf das werthöhere Byte der Adresse. Wegen der nun erforderlichen zweiten Addition dauert die Ausführung des Befehls vier Zyklen.

Den vom wertniedrigsten Byte adressierten Speicherbereich von 256 Bytes bezeichnet man als Seite (Page). Das werthöhere Byte der Adresse kann damit 256 verschiedene Seiten adressieren. Liefert die Berechnung der neuen Befehlsadresse einen Übertrag, so spricht man von einer Seiten-Überschreitung, da die alte und die neue Befehlsadresse in verschiedenen Seiten liegen.

Page

Beispiele:

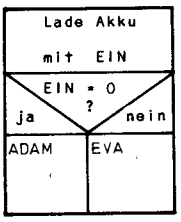
```
Befehlszähler 1121   Seite 11
  Sprungweite   6
 neue Adresse 1127   Seite 11

Befehlszähler 11F1   Seite 11
  Sprungweite  2A
 neue Adresse 121B   Seite 12  Seitenüberschreitung!
```

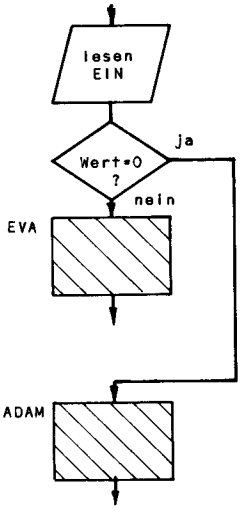
Bei der Anwendung eines bedingten Verzweigungsbefehls ist zunächst dafür zu sorgen, daß die Sprungbedingung, meist der Inhalt einer Speicherstelle, über einen Befehl das Bedingungsregister beeinflussen kann.

Beispiel:
Der Inhalt des Eingaberegisters EIN entscheide über den Ablauf des Programms.
Für EIN=0 Fortsetzung des Programms bei der Adresse ADAM .
Für EIN≠0 Fortsetzung des Programms bei der Adresse EVA .

Der Inhalt der Speicherstelle EIN hat keinen Einfluß auf das Bedingungsregister, erst der Transport in den Akkumulator bringt je nach Inhalt des übertragenen Bytes das Z-Bit auf 0 oder 1. Dann kann ein bedingter Verzweigungsbefehl das Z-Bit auswerten und entsprechend verzweigen.

Das Z-Bit (Zero = Null) des Bedingungsregisters wird dann durch einen Lade-Befehl auf 0 gesetzt, wenn das transportierte Byte nicht Null ist; besteht es jedoch aus acht Oer-Bits, so wird es 1 gesetzt.





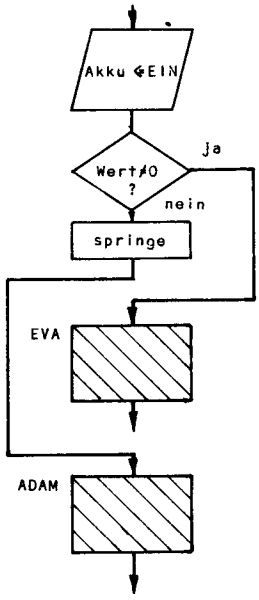
Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		EIN	=$1702	
			*=$0300	
0300	AD	START	LDA	EIN
	02			
	17			
0303	F0		BEQ	ADAM
	1B			
0305		EVA		
0320		ADAM		

Berechnung der Sprungweite

Hexadezimal Zieladresse	0320
Befehlszähler	-0305
relative Adresse	1B

Liegt die Zieladresse mehr als 127 Bytes entfernt, so muß der unbedingte Sprung JMP zu Hilfe genommen werden. Dabei kann es erforderlich sein, die Logik der Abfrage umzudrehen.

In dem folgenden Beispiel sei die symbolische Adresse ADAM mehr als 127 Bytes entfernt, das Sprungziel EVA liege in der Nähe. Die Abfrage auf ≠0 führt direkt nach EVA , der unbedingte Sprung geht nach ADAM.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		EIN	=$1702	
			*=$0300	
0300	AD	START	LDA	EIN
	02			
	17			
0303	D0		BNE	EVA
	03			
0305	4C		JMP	ADAM
	F0			
	03			
0308		EVA		
03F0		ADAM		

Berechnung der Sprungweite

Zieladresse	0308
Befehlszähler	0305
relative Adresse	03

4.3.2 Die bedingten Sprünge des 6800 **6800**

Alle bedingten Sprünge sind Verzweigungsbefehle, deren Wirkung bereits im Abschnitt 4.2.2. gezeigt wurde. Das erste Byte enthält den Funktionscode, das zweite die Sprungweite als relative Adresse.
Bei der symbolischen Programmierung steht im Operandenteil eine symbolische Sprungadresse, der Assembler berechnet die Sprungweite und setzt sie in das zweite Byte des Befehls ein.
Die bedingten Verzweigungsbefehle werten die Bits des Bedingungsregisters aus.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --Hi NZVC	
BCC	Adresse	relativ	24	2	4		Verzweige bei C=0 Ergebnis ohne Übertrag
BCS	Adresse	relativ	25	2	4		Verzweige bei C=1 Ergebnis mit Übertrag
BNE	Adresse	relativ	26	2	4		Verzweige bei Z=0 Ergebnis ungleich Null
BEQ	Adresse	relativ	27	2	4		Verzweige bei Z=1 Ergebnis gleich Null
BVC	Adresse	relativ	28	2	4		Verzweige bei V=0 Ergebnis ohne Überlauf
BVS	Adresse	relativ	29	2	4		Verzweige bei V=1 Ergebnis mit Überlauf
BPL	Adresse	relativ	2A	2	4		Verzweige bei N=0 Ergebnis positiv
BMI	Adresse	relativ	2B	2	4		Verzweige bei N=1 Ergebnis negativ

Der Buchstabe B ist eine Abkürzung für Branch = Verzweige.
Der Befehl BCC bedeutet Branch on Carry Clear = Verzweige, wenn das Carry-Bit gelöscht ist.

Die folgenden Verzweigungsbefehle untersuchen das Ergebnis einer Subtraktion; z.B. Akkumulator minus Vergleichswert für positive Dualzahlen ohne Vorzeichen.

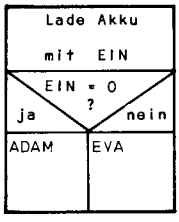
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI NZVC	
BCS	Adresse	relativ	25	2	4		Verzweige bei C=1 Akku < Operand
BLS	Adresse	relativ	23	2	4		Verzweige bei (C ODER Z)=1 Akku ≦ Operand
BEQ	Adresse	relativ	27	2	4		Verzweige bei Z=1 Akku = Operand
BCC	Adresse	relativ	24	2	4		Verzweige bei C=0 Akku ≧ Operand
BHI	Adresse	relativ	22	2	4		Verzweige bei (C UND Z)=0 Akku > Operand

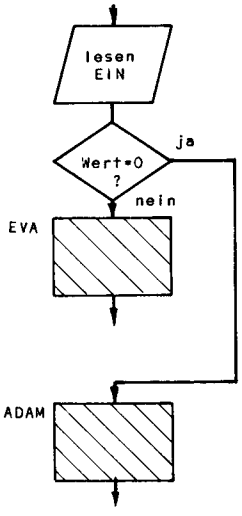
Die folgenden Vergleichsbefehls untersuchen das Ergebnis einer Subtraktion; z.B. Akkumulator minus Vergleichswert für Dualzahlen mit Vorzeichen im 2er Komplement.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI NZVC	
BLT	Adresse	relativ	2D	2	4		Verzweige bei (N XOR V)=1 Ak.< Op.
BLE	Adresse	relativ	2F	2	4		Verzweige bei (Z ODER (N XOR V))=1 Ak.≦ Op.
BEQ	Adresse	relativ	27	2	4		Verzweige bei Z=1 Ak.= Op.
BGE	Adresse	relativ	2C	2	4		Verzweige bei (N XOR V)=0 Ak.≧ Op.
BGT	Adresse	relativ	2E	2	4		Verzweige bei (Z ODER (N XOR V))=0 Ak.> Op.

Bei der Anwendung eines bedingten Verzweigungsbefehls ist zunächst dafür zu sorgen, daß die Sprungbedingung, meist der Inhalt einer Speicherstelle, über einen Befehl das Bedingungsregister beeinflussen kann.

Beispiel:
Der Inhalt des Eingaberegisters EIN entscheide über den Ablauf eines Programms.
Für EIN=0 Fortsetzung des Programms bei der Adresse ADAM.
Für EIN≠0 Fortsetzung des Programms bei der Adresse EVA .
Der Inhalt der Speicherstelle EIN muß dazu mit einem Lade-Befehl in einen Akkumulator gebracht werden. Dabei wird das Z-Bit (Zero = Null) des Bedingungsregisters je nach dem Wert des transportierten Bytes beeinflußt. Besteht es aus acht Oer-Bits, so wird Z = 1 ; ist der Wert nicht Null, so wird Z = 0 gesetzt.



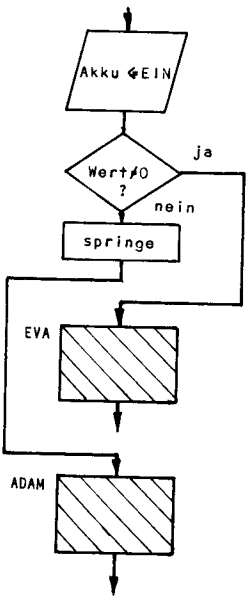


Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		EIN	EQU	$8006
			ORG	$0180
0180	86	START	LDA A	EIN
	80			
	06			
0183	27		BEQ	ADAM
	1B			
0185		EVA		
01A0		ADAM		

Berechnung der Sprungweite

Hexadezimal	Zieladresse 01A0
	Befehlszähler -0185
	relative Adresse 1B

Liegt die Zieladresse mehr als 127 Bytes entfernt, so muß der unbedingte Sprung JMP zu Hilfe genommen werden. Dabei kann es erforderlich sein, die Logik der Abfrage umzudrehen.
In dem folgenden Beispiel sei die symbolische Adresse ADAM mehr als 127 Bytes entfernt, das Sprungziel EVA liege in der Nähe. Die Abfrage auf ≠ 0 führt direkt nach EVA, der unbedingte Sprung geht nach ADAM.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		EIN	EQU	$8006
			ORG	$0180
0180	B6	START	LDA A	EIN
	80			
	06			
0183	26		BNE	EVA
	03			
0185	7E		JMP	ADAM
	01			
	F0			
0188		EVA		
01F0		ADAM		

Berechnung der Sprungweite

Hexadezimal	Zieladresse	0188
	Befehlszähler	-0185
	relative Adresse	03

4.3.3 Die bedingten Sprünge des 8080/8085

8080

Alle bedingten Sprünge enthalten eine zwei byte lange Adresse des Sprungziels. Die wertniedrigsten Bits stehen im zweiten Byte, die werthöchsten im dritten Byte des Befehls. Das Befehlszählregister wird nur dann mit der neuen Adresse überschrieben, wenn die Sprungbedingung erfüllt ist, anderenfalls wird der nächste Befehl ausgeführt.
In der Assemblerschreibweise steht im Operandenfeld die symbolische Zieladresse. Die bedingten Sprungbefehle werten die Bits des Bedingungsregisters aus.

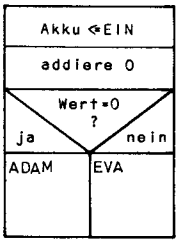
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H -P-C	
JC	Adresse	absolut	DA	3	10		Springe bei C=1 Ergebnis mit Übertrag
JNC	Adresse	absolut	D2	3	10		Springe bei C=0 Ergebnis ohne Übertrag
JZ	Adresse	absolut	CA	3	10		Springe bei Z=1 Ergebnis gleich Null
JNZ	Adresse	absolut	C2	3	10		Springe bei Z=0 Ergebnis ungleich Null
JM	Adresse	absolut	FA	3	10		Springe bei N=1 Ergebnis negativ (minus)
JP	Adresse	absolut	F2	3	10		Springe bei N=0 Ergebnis positiv
JPE	Adresse	absolut	EA	3	10		Springe bei P=1 Parität gerade
JPO	Adresse	absolut	E2	3	10		Springe bei P=0 Parität ungerade

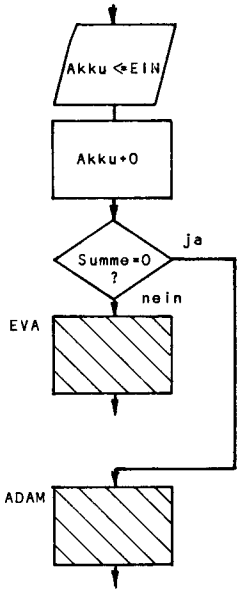
Der Buchstabe J bedeutet Jump = Springe; die Buchstaben dahinter geben die Sprungbedingung an.
JC bedeutet Jump on Carry = Springe, wenn das Carry-Bit gesetzt ist.

Vor einem bedingten Sprung muß das Bedingungsregister entsprechend der Sprungbedingung beeinflußt werden. Transportbefehle (LDA, MOV, IN) verändern nicht die Bedingungsbits! Durch Addition der Konstanten 00 (Befehl ADI 00H) bzw. das logische ODER des Akkumulators mit sich selbst (Befehl ORA A) werden die Bedingungsbits beeinflußt,ohne den Inhalt des Akkumulators zu verändern.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
ADI	konst	unmittelbar	C6	2	7	XX X X X	Addiere Akkumulator mit 1-Byte-Konstanten
ORA	A	code-eigen	B7	1	4	XX X X X	Akku ← Akku ODER Akku

Beispiel:
Der Inhalt des Eingaberegisters 01 entscheide über den Ablauf eines Programms.
Ist der Wert = 0, so soll das Programm bei ADAM fortfahren; ist der Wert ≠ 0, so soll das Programm bei EVA fortfahren.
Dazu wird der Wert in den Akkumulator geladen und mit der Konstanten 0 addiert, um das Z-Bit (Zero = Null) zu beeinflussen.





Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	DB	START	IN	01H
	01			
2882	C6		ADI	00H
	00			
2884	CA		JZ	ADAM
	F0			
	28			
2887		EVA		
28F0		ADAM		

Übungen zum Abschnitt 4.2. und 4.3. **A**

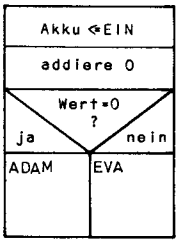
Für einen Test der Programme mit einer Schaltung nach Bild 48 muß vorher die Parallelschnittstelle programmiert werden.

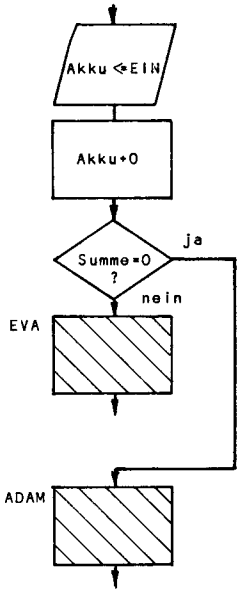
Der Buchstabe J bedeutet Jump = Springe; die Buchstaben dahinter geben die Sprungbedingung an.
JC bedeutet Jump on Carry = Springe, wenn das Carry-Bit gesetzt ist.

Vor einem bedingten Sprung muß das Bedingungsregister entsprechend der Sprungbedingung beeinflußt werden. Transportbefehle (LDA, MOV, IN) verändern nicht die Bedingungsbits! Durch Addition der Konstanten 00 (Befehl ADI 00H) bzw. das logische ODER des Akkumulators mit sich selbst (Befehl ORA A) werden die Bedingungsbits beeinflußt,ohne den Inhalt des Akkumulators zu verändern.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
ADI	konst	unmittelbar	C6	2	7	XX X X X	Addiere Akkumulator mit 1-Byte-Konstanten
ORA	A	code-eigen	B7	1	4	XX X X X	Akku ← Akku ODER Akku

Beispiel:
Der Inhalt des Eingaberegisters 01 entscheide über den Ablauf eines Programms.
Ist der Wert = 0, so soll das Programm bei ADAM fortfahren; ist der Wert ≠ 0, so soll das Programm bei EVA fortfahren.
Dazu wird der Wert in den Akkumulator geladen und mit der Konstanten 0 addiert, um das Z-Bit (Zero = Null) zu beeinflussen.





Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	DB	START	IN	01H
	01			
2882	C6		ADI	00H
	00			
2884	CA		JZ	ADAM
	F0			
	28			
2887		EVA		
28F0		ADAM		

Übungen zum Abschnitt 4.2. und 4.3. **A**

Für einen Test der Programme mit einer Schaltung nach Bild 48 muß vorher die Parallelschnittstelle programmiert werden.

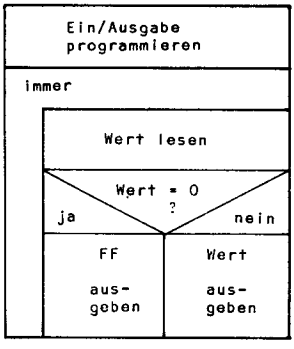
1.Aufgabe
In einer unendlichen Schleife ist der Inhalt des Eingaberegisters EIN in das Ausgaberegister AUS zu bringen.Ist der Eingabewert 00, so soll FF angezeigt werden.

2.Aufgabe
In einer unendlichen Schleife ist der Inhalt des Eingaberegisters EIN zu untersuchen.
a. Für EIN=0 soll das Programm in eine Warteschleife verzweigen, die keine Eingabe mehr annimmt und konstant 00 anzeigt.
b. Für EIN positiv (Bit 7 = 0) ist der eingegebene Wert ohne Änderung wieder auszugeben.
c. Für EIN negativ (Bit 7 = 1) ist die Konstante FF auszugeben.

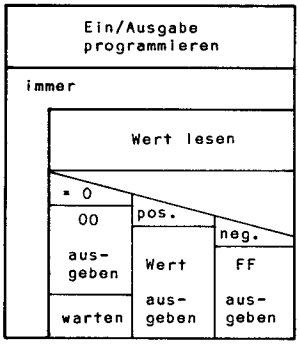
3.Aufgabe
In einer unendlichen Schleife ist der Inhalt des Eingaberegisters EIN zu untersuchen.
a. Ist der Inhalt gleich Null, so ist das Bitmuster 11111111 = FF anzuzeigen.
b. Ist der Inhalt positiv (Bit 7 = 0), so ist das Bitmuster 10011001 = 99 anzuzeigen.
c. Ist der Inhalt negativ (Bit 7 = 1), so ist das Bitmuster 01100110 = 66 anzuzeigen.

Programmblockpläne

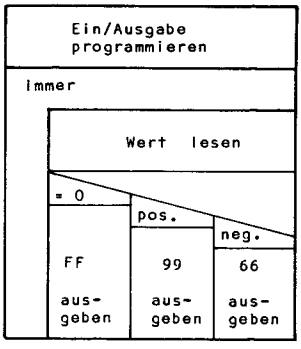
1.Aufgabe



2.Aufgabe



3.Aufgabe



4.4 Zählbefehle und Zählschleifen

Zählen heißt, eine Speicherstelle (Register oder Datenbyte) mit einem Anfangswert zu laden und dann fortlaufend um 1 zu erhöhen (aufwärts zählen) oder um 1 zu vermindern (abwärts zählen).Die Mikroprozessoren enthalten besondere Zählbefehle, die mit der Schrittweite 1 zählen; für größere Schrittweiten können Additions- bzw. Subtraktionsbefehle verwendet werden.

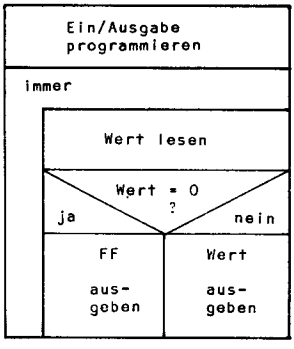
1.Aufgabe
In einer unendlichen Schleife ist der Inhalt des Eingaberegisters EIN in das Ausgaberegister AUS zu bringen.Ist der Eingabewert 00, so soll FF angezeigt werden.

2.Aufgabe
In einer unendlichen Schleife ist der Inhalt des Eingaberegisters EIN zu untersuchen.
a. Für EIN=0 soll das Programm in eine Warteschleife verzweigen, die keine Eingabe mehr annimmt und konstant 00 anzeigt.
b. Für EIN positiv (Bit 7 = 0) ist der eingegebene Wert ohne Änderung wieder auszugeben.
c. Für EIN negativ (Bit 7 = 1) ist die Konstante FF auszugeben.

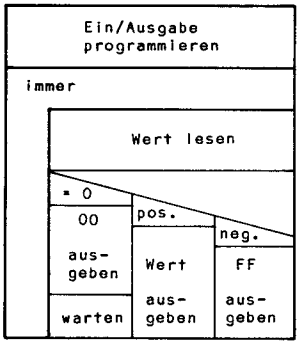
3.Aufgabe
In einer unendlichen Schleife ist der Inhalt des Eingaberegisters EIN zu untersuchen.
a. Ist der Inhalt gleich Null, so ist das Bitmuster 11111111 = FF anzuzeigen.
b. Ist der Inhalt positiv (Bit 7 = 0), so ist das Bitmuster 10011001 = 99 anzuzeigen.
c. Ist der Inhalt negativ (Bit 7 = 1), so ist das Bitmuster 01100110 = 66 anzuzeigen.

Programmblockpläne

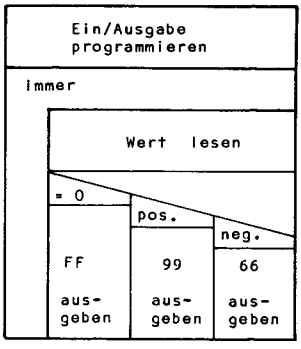
1.Aufgabe



2.Aufgabe



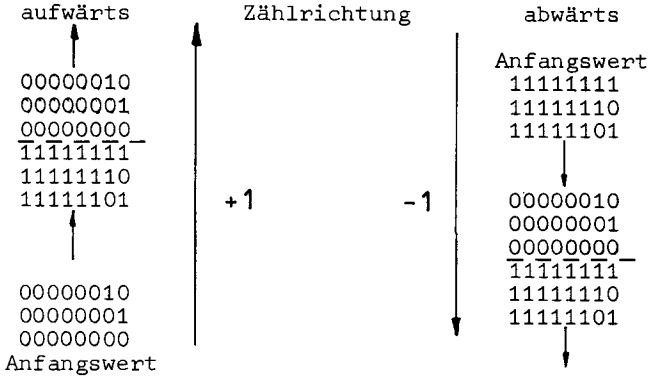
3.Aufgabe



4.4 Zählbefehle und Zählschleifen

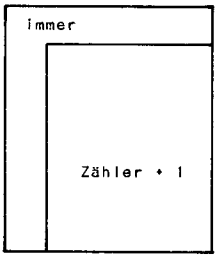
Zählen heißt, eine Speicherstelle (Register oder Datenbyte) mit einem Anfangswert zu laden und dann fortlaufend um 1 zu erhöhen (aufwärts zählen) oder um 1 zu vermindern (abwärts zählen).Die Mikroprozessoren enthalten besondere Zählbefehle, die mit der Schrittweite 1 zählen; für größere Schrittweiten können Additions- bzw. Subtraktionsbefehle verwendet werden.

Freilaufende Zähler haben keine Ende-Bedingung. Ein 8-Bit-Aufwärtszähler beginnt nach dem Erreichen des größten Wertes von 11111111 wieder bei 00000000, ein entsprechender Abwärtszähler läuft nach dem Wert 00000000 mit 11111111 weiter abwärts.

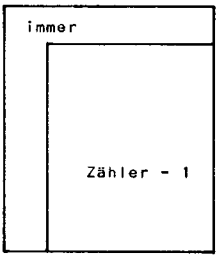


Programmblockplan

Aufwärtszähler

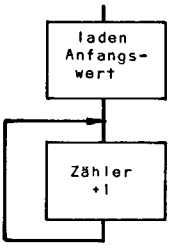


Abwärtszähler

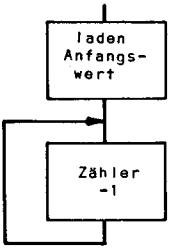


Programmablaufplan

Aufwärtszähler



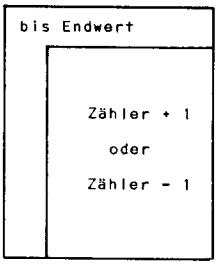
Abwärtszähler



Einstellbare Zähler werden durch bedingte Sprünge kontrolliert und nach dem Erreichen eines bestimmten Zählerstandes abgebrochen (Zählschleifen).

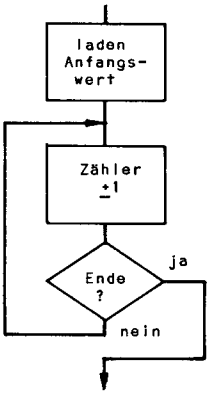
Programmblockplan

Aufwärts- bzw. Abwärtszähler

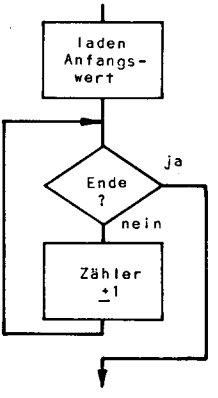


Programmablaufpläne

Abfrage am Schleifenende



Abfrage am Schleifenanfang

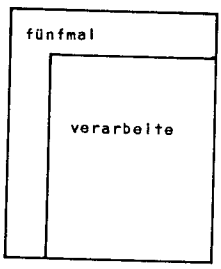


Der bedingte Sprung, der die Schleife beendet, kann vor oder hinter dem Zählbefehl liegen.
Die Abfrage am Schleifenende ist übersichtlicher und bei Schleifen mit einer konstanten Zahl von Durchläufen zu bevorzugen, wenn z.B. der Zähler mit einem Direktoperanden geladen wird.
Die Abfrage am Schleifenanfang ist bei einer variablen Zahl von Durchläufen zu bevorzugen, wenn z.B. der Anfangswert von außen eingegeben wird. Für den Fall, daß der eingegebene Anfangswert gleich dem vorgegebenen Endwert ist, wird die Schleife sofort abgebrochen. Bei einer Abfrage nach dem Erhöhen des Zählers würde der Zähler einmal voll durchlaufen.

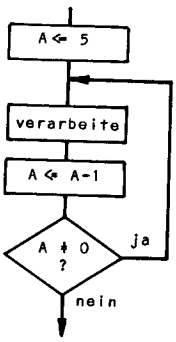
Die Zählbefehle beeinflussen in jedem Fall das Z-Bit (Nullbedingung) des Bedingungsregisters. Für den Fall, daß der laufende Zählerstand nicht zur Verarbeitung benötigt wird, kann man einen Abwärtszähler bis Null laufen lassen.

Beispiel: Eine Schleife soll fünfmal durchlaufen werden, ein Abwärtszähler läuft von 5 bis 0 im Akkumulator, der laufende Zählerstand wird nicht benötigt.

Programmblockplan



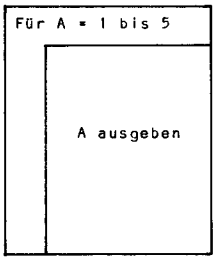
Programmablaufplan

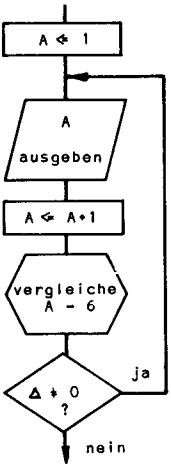


Bei vielen Anwendungen (z. B. Adreßrechnungen) benötigt man jedoch den laufenden Zählerstand zur Verarbeitung. In diesem Fall muß oft ein Aufwärtszähler auf einen Endwert kontrolliert werden. Dies geschieht durch Vergleichsbefehle, die die Differenz zwischen einem Registerinhalt und einem Vergleichswert bilden und das Bedingungsregister beeinflussen, ohne jedoch den Zählerstand zu verändern.

Beispiel: Ein Zähler laufe von 1 bis 5, der laufende Zählerstand werde ausgegeben. Fragt man am Ende der Schleife mit dem Befehl "Springe bei ungleich Null" ab, so muß der Vergleichswert um 1 größer sein als der Endwert, damit der Endwert noch verarbeitet werden kann. Beim Befehl "Springe bei negativ" wäre mit dem Endwert zu vergleichen, die Schleife wird nach dem Nulldurchgang der Differenz verlassen. Von diesem Befehl ist jedoch abzuraten, da alle Differenzen größer 7F für den Mikroprozessor negative Zahlen sind und das N-Bit 1 setzen.

Programmblockplan



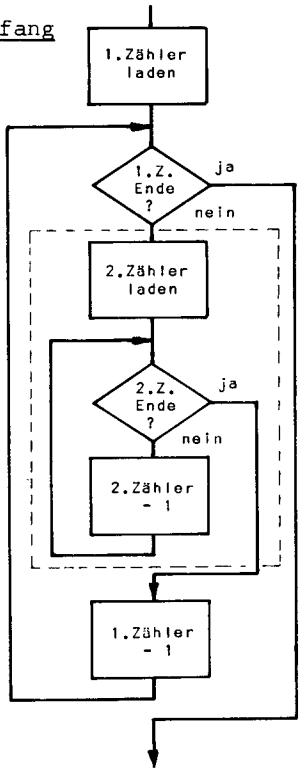


Programmablaufplan

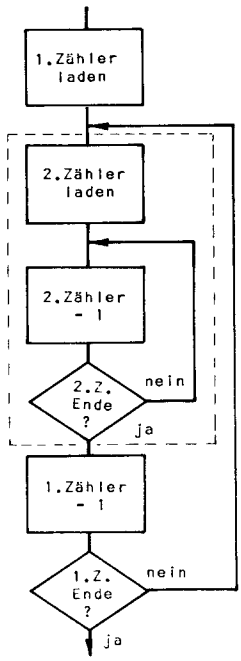
Sieht man den Zählerinhalt als positive Dualzahl an, so läuft ein 8-Bit-Zähler von 0 bis 255, ein 16-Bit-Zähler von 0 bis 65 535. Bei Verzögerungsschleifen reicht oft die mit einem Durchlauf erreichte Zeitverzögerung nicht aus; durch eine zweite äußere Schleife kann der innere Zähler mehrmals durchlaufen werden. Bei geschachtelten Schleifen multiplizieren sich die Einzeldurchläufe.

Beispiel: Die beiden folgenden Programmablaufpläne zeigen zwei geschachtelte Abwärtszähler, einmal mit der Abfrage am Anfang

Abfrage am Schleifenanfang



Abfrage am Schleifenende



(z.B. bei Variablen), einmal mit der Abfrage am Ende der Schleife (z.B. bei Konstanten). Der Programmblockplan zeigt nur die Laufbedingung oder die Zahl der Durchläufe; er ist wesentlich kürzer und übersichtlicher.

4.4.1 Die Zähl- und Vergleichsbefehle des KIM-1 KIM

Die folgenden Zählbefehle erhöhen bzw. erniedrigen den Inhalt eines Zählers um 1. Sie lassen sich sowohl auf Register als auch auf Datenspeicherstellen anwenden. Der neue Zählerstand beeinflußt das N- und Z-Bit des Bedingungsregisters.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	V	B	D	I	Z	C	
INX		code-eigen	E8	1	2	X					X		Erhöhe X-Register um 1
INY		code-eigen	C8	1	2	X					X		Erhöhe Y-Register um 1
INC	Adresse	absolut	EE	3	6	X					X		Erhöhe Speicherstelle um 1
INCZ	Adresse	zero-page	E6	2	5	X					X		Erhöhe Speicherstelle um 1
DEX		code-eigen	CA	1	2	X					X		Vermindere X-Register um 1
DEY		code-eigen	88	1	2	X					X		Vermindere Y-Register um 1
DEC	Adresse	absolut	CE	3	6	X					X		Vermindere Speicherstelle um 1
DECZ	Adresse	zero-page	C6	2	5	X					X		Vermindere Speicherstelle um 1

IN ist eine Abkürzung für INcrementiere = erhöhe;
DE ist eine Abkürzung für DEcrementiere = vermindere.

a. Abwärtszähler mit Nullabfrage

Beispiel: Im X-Register ist ein Abwärtszähler mit der größtmöglichen Zahl von Durchläufen aufzubauen, beim Zählerstand 0 soll beim Befehl MAX fortgefahren werden. Die Ausführungszeit dieser Schleife ist zu berechnen. Da der Anfangswert konstant FF ist, wird die Abfrage an das Ende der Schleife gelegt.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand	
			*=$0300		
0300	A2	START	LDX	#$FF	2 ZYKLEN
	FF				
0302	CA	OTTO	DEX		2 ZYKLEN
0303	D0		BNE	OTTO	3 ZYKLEN
	FD				
0305		MAX			

Berechnung der Ausführungszeit:

Schleife	255 X 5	=	1275 Zyklen
Laden mit	FF	=	2 Zyklen
Zusammen		=	1277 Zyklen

Beginnt man mit dem Ausgangswert 00 und fragt wieder am Ende der Schleife ab, so lassen sich 256 Durchläufe erzielen.

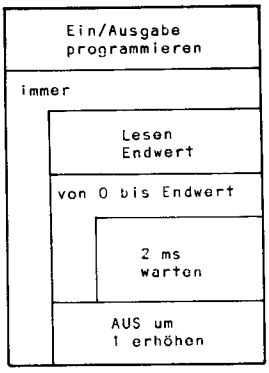
Bei einer Zykluszeit von 1µs liefert diese Schleife eine Verzögerung von 1,277 ms. Längere Zeitverzögerungen lassen sich durch eine zweite außen liegende Schleife oder durch eine Verlängerung der bestehenden Schleife erreichen. Dazu kann der NOP-Befehl dienen, der eine Zeitverzögerung von zwei Zyklen

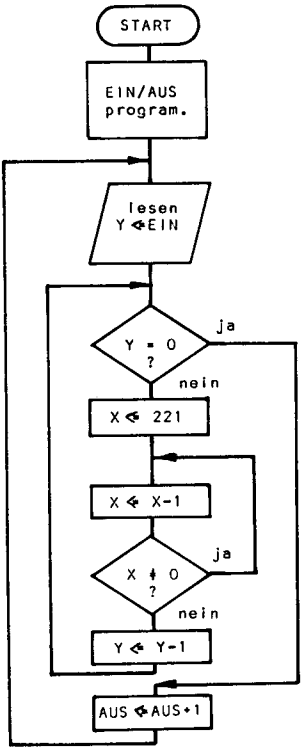
bringt, aber keine Operationen ausführt. Er wird auch oft dazu benutzt, in einem Programm Platz für noch einzubauende Befehle freizuhalten oder Lücken nicht mehr benötigter Befehle auszufüllen. Bei zeitkritischen Programmen ist es jedoch vorteilhafter, größere Lücken zu überspringen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	
NOP		code-eigen	EA	1	2		Keine Operation, Erhöhung des Befehlszähl.

NOP bedeutet No OPeration = keine Operation.

Das folgende Beispiel enthält zwei Zählschleifen. Eine innere Schleife im X-Register soll konstant um 2 ms verzögern, eine äußere Schleife soll im Y-Register laufen. Als Ausgangswert dient der Inhalt der Speicherstelle EIN, der von außen über Kippschalter entsprechend Bild 48 eingegeben wird. Da hierbei auch der Wert Null vorkommen kann, liegt die Abfrage der äußeren Schleife am Anfang. Die Schalterstellung gibt an, wie oft um den Faktor 2 ms verzögert werden soll. Nach Ablauf der beiden Schleifen soll eine mit einer Anzeigeschaltung verbundene Speicherstelle AUS um 1 erhöht werden. Die Schaltung nach Bild 48 läuft damit als Zähler, dessen Zählfrequenz über Kippschalter einstellbar ist.





```
Hexadezimal                    Symbolisch
Adresse Inhalt   Name   Befehl    Operand

                 AUS    =$1700
                 RAU    =$1701
                 EIN    =$1702
                 REI    =$1703

                        *=$0300
0300    A9       START  LDA       #$00
        00
0302    8D              STA       REI
        03
        17
0305    A9              LDA       #$FF
        FF
0307    8D              STA       RAU
        01
        17
030A    AC       MAUS   LDY       EIN
        02
        17
030D    F0       SUSI   BEQ       MAX
        0B
030F    A2              LDX       #$DD
        DD
0311    CA       OTTO   DEX
0312    EA              NOP
0312    EA              NOP
0314    D0              BNE       OTTO
        FB
0316    88              DEY
0317    4C              JMP       SUSI
        0D
        03
031A    EE       MAX    INC       AUS
        00
        17
031D    4C              JMP       MAUS
        0A
        03
                        .END
```

Verzögerung der inneren Schleife:

Schleife	221 X 9 =	1989 Zyklen
Laden mit DD_{16} = 221_{10}	=	2 Zyklen
Abfrage äußere Schleife	=	3 Zyklen
Vermindern Zähler	=	2 Zyklen
unbedingter Sprung	=	3 Zyklen
		1999 Zyklen

b. Aufwärtszähler mit Abfrage auf Endwert

Schleifen, die Adreßrechnungen für Indexschleifen durchführen müssen, zählen meist von Null aufwärts und müssen auf einen Endwert abgefragt werden. Da die Verzweigungsbefehle nur auf Null abfragen, muß die Differenz zwischen dem laufenden Wert und dem

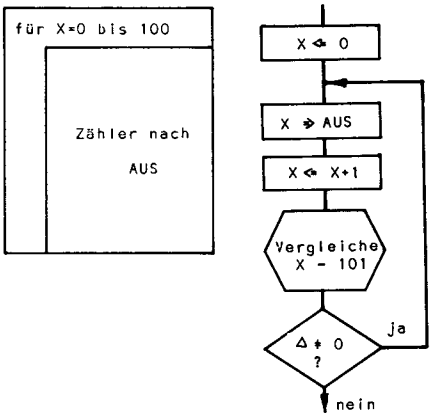
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	V	B	D	I	Z	C	
CMP	Adresse	absolut	CD	3	4	X					X	X	Vergleiche Akkumulator (absolut adressiert)
CMPZ	Adresse	zero-page	C5	2	3	X					X	X	Vergleiche Akkumulator (zero-page adressiert)
CMP	# konst	unmittelbar	C9	2	2	X					X	X	Vergleiche Akkumulator mit Konstanten
CPX	Adresse	absolut	EC	3	4	X					X	X	Vergleiche X-Register (absolut adressiert)
CPXZ	Adresse	zero-page	E4	2	3	X					X	X	Vergleiche X-Register (zero-page adressiert)
CPX	# konst	unmittelbar	E0	2	2	X					X	X	Vergleiche X-Register mit Konstanten
CPY	Adresse	absolut	CC	3	4	X					X	X	Vergleiche Y-Register (absolut adressiert)
CPYZ	Adresse	zero-page	C4	2	3	X					X	X	Vergleiche Y-Register (zero-page adressiert)
CPY	# konst	unmittelbar	C0	2	2	X					X	X	Vergleiche Y-Register mit Konstanten

Endwert gebildet werden. Die folgenden Vergleichsbefehle subtrahieren vom Inhalt eines Registers einen Vergleichswert und beeinflussen damit das Bedingungsregister, ohne jedoch die Differenz in das Register zu übernehmen. Der laufende Zählerstand bleibt damit erhalten!

CMP bedeutet COmPare = vergleiche.

Mit Hilfe dieser Vergleichsbefehle können Schleifen gebildet werden, die bis zu einem Endwert laufen.

Beispiel: Im X-Register ist ein Aufwärtszähler von 0 bis 100 aufzubauen, dessen Zählerstand laufend zur Speicherstelle AUS zu bringen ist. Da der Endwert noch zu verarbeiten ist, wurde hier der Vergleichswert um 1 höher angenommen. $101_{10} = 65_{16}$



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	=$1700	
			*=$0300	
0300	A2		LDX	#$00
	00			
0302	8E	SUS1	STX	AUS
	00			
	17			
0305	E8		INX	
0306	E0		CPX	#$65
	65			
0308	D0		BNE	SUS1
	F8			
030A				

4.4.2 Die Zähl- und Vergleichsbefehle des 6800 **6800**

Die folgenden Zählbefehle erhöhen bzw. erniedrigen den Inhalt eines Zählers um 1. Sie lassen sich sowohl auf die 8-Bit-Register (Akkumulator A und B) als auch auf die 16-Bit-Register (Indexregister und Stapelzeiger) und auch auf 8-Bit-Speicherstellen anwenden. Das Ergebnis beeinflußt das Bedingungsregister. Das V-Bit zeigt den Überlauf in der 2er-Komplement-Darstellung an. Bei Befehlen, die um 1 erhöhen, wird das V-Bit dann 1 gesetzt

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen ↓↓HI NZVC	
INC A		code-eigen	4C	1	2	XXX	Erhöhe Akkumulator A um 1
INC B		code-eigen	5C	1	2	XXX	Erhöhe Akkumulator B um 1
INX		code-eigen	08	1	4	X	Erhöhe Indexregister um 1
INS		code-eigen	31	1	4		Erhöhe Stapelzeiger um 1
INC	Adresse	absolut	7C	3	6	XXX	Erhöhe Speicherstelle um 1
DEC A		code-eigen	4A	1	2	XXX	Vermindere Akkumulator A um 1
DEC B		code-eigen	5A	1	2	XXX	Vermindere Akkumulator B um 1
DEX		code-eigen	09	1	4	X	Vermindere Indexregister um 1
DES		code-eigen	34	1	4		Vermindere Stapelzeiger um 1
DEC	Adresse	absolut	7A	3	6	XXX	Vermindere Speicherstelle um 1

wenn der Zählerstand vorher $7F_{16}$ war. Bei Befehlen, die um 1 erniedrigen, wird das V-Bit dann gesetzt, wenn der Zählerstand vorher 80_{16} war.

INC bedeutet INCrementiere = erhöhe. DEC bedeutet DECrementiere = vermindere.

Der Stapelzeiger eignet sich nicht zum Aufbau von Schleifen, er sollte nur für die Stapeladressierung verwendet werden.

a. Abwärtszähler mit Nullabfrage

Beispiel: Im Akkumulator A ist ein Abwärtszähler mit der größtmöglichen Zahl von Durchläufen (Ausgangswert FF) aufzubauen; beim Zählerstand Null soll anschließend im Indexregister ebenfalls ein Abwärtszähler mit dem Ausgangswert FFFF ablaufen; beim Zählerstand Null soll das Programm mit dem Befehl MAX fortfahren. Die Ausführungszeit der Schleifen hängt vom Taktgenerator des verwendeten Mikrocomputers ab. In allen folgenden Beispielen wird eine Taktfrequenz von 614,4 kHz entsprechend einer Zykluszeit von 1,6276 µs zugrunde gelegt. Da die Ausgangswerte konstant sind, wird die Abfrage an das Ende der Schleife gelegt.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand	
			ORG	$0180	
0180	86		LDA A	#$FF	2 ZYKLEN
	FF				
0182	4A	PAUL	DEC A		2 ZYKLEN
0183	26		BNE	PAUL	4 ZYKLEN
	FD				
0185	CE		LDX	#$FFFF	3 ZYKLEN
	FF				
	FF				
0188	09	OPA	DEX		4 ZYKLEN
0189	26		BNE	OPA	4 ZYKLEN
	FD				
018B		MAX			

Akkumulator-Schleife

Schleife	255 X 6	= 1530 Zyklen
Laden mit FF	=	2 Zyklen
zusammen	=	1532 Zyklen

Bei einer Zykluszeit von 1,6276 µs liefert diese Schleife eine Verzögerung von 2,4935 ms. Beginnt man mit dem Ausgangswert 00 und fragt wieder am Ende der Schleife ab, so lassen sich 256 Durchläufe erzielen.

Indexregister-Schleife

Schleife	65535 X 8	= 524280 Zyklen
Laden mit FFFF	=	3 Zyklen
zusammen	=	524283 Zyklen

Bei einer Zykluszeit von 1,6276 µs liefert diese Schleife eine Verzögerung von 853,318 ms. Beginnt man mit dem Ausgangswert 0000 und fragt wieder am Ende der Schleife ab, so lassen sich 65536 Durchläufe erzielen.

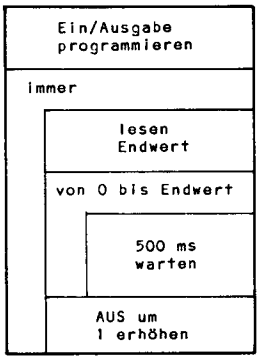
Weitere Zeitverzögerungen lassen sich durch eine zweite äußere Schleife oder eine Verlängerung der bestehenden Schleife erreichen. Dazu kann der NOP-Befehl dienen, der eine Zeitverzögerung

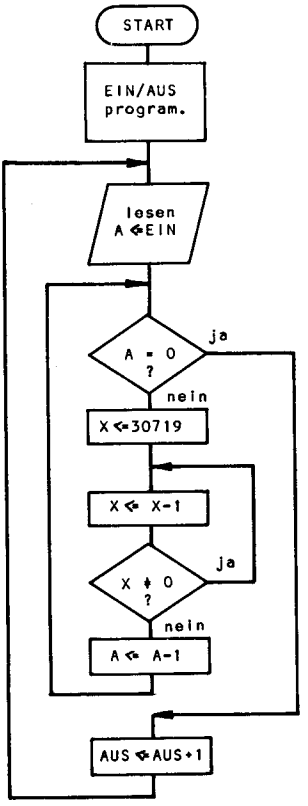
von zwei Zyklen bringt, aber keine Operation ausführt. Er wird auch oft dazu benutzt, in einem Programm Platz für noch einzubauende Befehle frei zu halten oder Lücken nicht mehr benötigter Befehle auszufüllen. Größere Lücken kann man auch einfach überspringen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI NZVC	
NOP		code-eigen	01	1	2		Keine Operation, Erhöhung des Befehlszähl.

NOP bedeutet No OPeration = keine Operation.

Das folgende Beispiel enthält zwei geschachtelte Zählschleifen. Eine innere Schleife im Indexregister soll konstant um 500 ms verzögern, eine äußere Schleife im Akkumulator A wird mit einem Ausgangswert geladen, der über Kippschalter einzugeben ist. Die Gesamtlaufzeit ist das Produkt der Laufzeiten beider Schleifen. Danach soll die Speicherstelle AUS um 1 erhöht werden, anschließend ist der Akkumulator A mit einem neuen Anfangswert zu laden. Verwendet man eine Schaltung nach Bild 48, so kann man über die Kippschalter verschiedene Laufzeitfaktoren eingeben und das Zählen der Speicherstelle AUS mit den Leuchtdioden verfolgen.





Verzögerung der inneren Schleife:

Schleife	30719 X 10	=	307190 Zyklen
Laden mit	77FF=30719	=	3 Zyklen
Abfrage auf Ende		=	4 Zyklen
vermindern um 1		=	2 Zyklen
unbedingter Sprung		=	3 Zyklen
			307202 Zyklen X 1,6276 µs ≈ 500 ms

```
Hexadezimal                Symbolisch
Adresse Inhalt   Name      Befehl     Operand

                 AUS       EQU        $8004
                 KONA      EQU        $8005
                 EIN       EQU        $8006
                 KONB      EQU        $8007

                           ORG        $0180
0180     7F      START     CLR        KONA
         80
         05
0183     86                LDA A      #$FF
         FF
0185     B7                STA A      AUS
         80
         04
0188     C6                LDA B      #$04
         04
018A     F7                STA B      KONA
         80
         05
018D     7F                CLR        KONB
         80
         07
0190     7F                CLR        EIN
         80
         06
0193     F7                STA B      KONB
         80
         07
0196     B6      SUSI      LDA A      EIN
         80
         06
0199     27      OTTO      BEQ        MAX
         0B
019B     CE                LDX        #$77FF
         77
         FF
019E     09      MAUS      DEX
019F     01                NOP
01A0     26                BNE        MAUS
         FC
01A2     4A                DEC A
01A3     7E                JMP        OTTO
         01
         99
01A6     7C      MAX       INC        AUS
         80
         04
01A9     7E                JMP        SUSI
         01
         96
                           END
```

b. Aufwärtszähler mit Abfrage auf Endwert

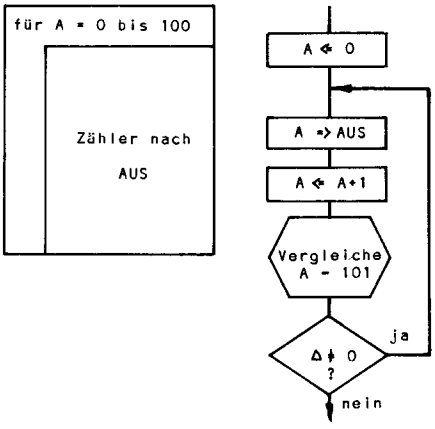
Schleifen, die Adreßrechnungen für Indexregister durchführen, zählen meist von einer Anfangsadresse aufwärts bis zu einer Endadresse. Verwendet man zur Abfrage einen Verzweigungsbefehl, so muß die Differenz zwischen dem laufenden Wert und dem Endwert gebildet werden. Die folgenden Vergleichsbefehle subtrahieren vom Inhalt eines Registers einen Vergleichswert und beeinflussen mit der Differenz das Bedingungsregister, ohne jedoch wie bei den Subtraktionsbefehlen das Ergebnis in das Register zu übernehmen. Der laufende Zählerstand bleibt erhalten. Die Vergleichsbefehle des Indexregisters sollen laut Herstellerangabe nur für den Vergleich auf Null verwendet werden.

CMP bedeutet CoMPare = vergleiche.

Mit Hilfe dieser Vergleichsbefehle können Schleifen gebildet werden, die bis zu einem bestimmten Endwert laufen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI	NZVC	
CBA		code-eigen	11	1	2		XXXX	Vergleiche Akku A - Akku B
CMP A	✚ konst	unmittelbar	81	2	2		XXXX	Vergleiche Akku A - 1-Byte-Konstante
CMP A	Adresse	absolut	B1	3	4		XXXX	Vergleiche Akku A - Speicher (extended)
CMPZ A	Adresse	zero-page	91	2	3		XXXX	Vergleiche Akku A - Speicher (direct)
CMP B	✚ konst	unmittelbar	C1	2	2		XXXX	Vergleiche Akku B - 1-Byte-Konstante
CMP B	Adresse	absolut	F1	3	4		XXXX	Vergleiche Akku B - Speicher (extended)
CMPZ B	Adresse	zero-page	D1	2	3		XXXX	Vergleiche Akku B - Speicher (direct)
CPX	✚ konst	unmittelbar	8C	3	3		XXX	Vergleiche X-Reg. - 2-Byte-Konstante
CPX	Adresse	absolut	BC	3	5		XXX	Vergleiche X-Reg. - Speicher (extended)
CPXZ	Adresse	zero-page	9C	2	4		XXX	Vergleiche X-Reg. - Speicher (direct)

Beispiel: Im Akkumulator A ist ein Aufwärtszähler von 0 bis 100 aufzubauen, dessen Zählerstand laufend zur Speicherstelle AUS zu bringen ist. Die Abfrage auf das Ende der Schleife übernimmt der Befehl "Verzweige bei ungleich Null". Da der Zähler nach der Ausgabe des laufenden Wertes erhöht wird, wird mit dem Endwert $100 + 1 = 101_{10} = 65_{16}$ verglichen, damit der Endwert 100 noch verarbeitet werden kann.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	EQU	$8004
			ORG	$0180
0180	4F	START	CLR A	
0181	B7	SUSI	STA A	AUS
	80			
	04			
0184	4C		INC A	
0185	81		CMP A	#$65
	65			
0187	26		BNE	SUSI
	F8			
0189		MAX		

4.4.3 Die Zähl- und Vergleichsbefehle des 8080/8085 **8080**

Dieser Abschnitt enthält Zählschleifen. Zur Berechnung der Ausführungszeit benötigt man für die Befehle der Schleife die Zahl der Taktzyklen. Die in den vorliegenden Befehlslisten angegebenen Werte gelten nicht für alle Ausführungen der Prozessoren 8080/8085!

Die folgenden Zählbefehle erhöhen bzw. erniedrigen den Inhalt eines 8-Bit-Registers um 1. Sie beeinflussen das Bedingungsregister.

INR bedeutet INcrementiere Register = erhöhe Register,
DCR bedeutet DeCRementiere Register = vermindere Register.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
INR	A	code-eigen	3C	1	5	XX X X	Erhöhe Akkumulator um 1
INR	B	code-eigen	04	1	5	XX X X	Erhöhe B-Register um 1
INR	C	code-eigen	0C	1	5	XX X X	Erhöhe C-Register um 1
INR	D	code-eigen	14	1	5	XX X X	Erhöhe D-Register um 1
INR	E	code-eigen	1C	1	5	XX X X	Erhöhe E-Register um 1
INR	H	code-eigen	24	1	5	XX X X	Erhöhe H-Register um 1
INR	L	code-eigen	2C	1	5	XX X X	Erhöhe L-Register um 1
DCR	A	code-eigen	3D	1	5	XX X X	Vermindere Akkumulator um 1
DCR	B	code-eigen	05	1	5	XX X X	Vermindere B-Register um 1
DCR	C	code-eigen	0D	1	5	XX X X	Vermindere C-Register um 1
DCR	D	code-eigen	15	1	5	XX X X	Vermindere D-Register um 1
DCR	E	code-eigen	1D	1	5	XX X X	Vermindere E-Register um 1
DCR	H	code-eigen	25	1	5	XX X X	Vermindere H-Register um 1
DCR	L	code-eigen	2D	1	5	XX X X	Vermindere L-Register um 1

a. Abwärtszähler mit Nullabfrage

Beispiel: Im Akkumulator ist ein Abwärtszähler mit der größtmöglichen Zahl von Durchläufen (Ausgangswert FF) aufzubauen, beim Zählerstand Null soll das Programm mit dem Befehl MAX fortfahren. Die Ausführungszeit ist vom verwendeten Taktgenerator abhängig. In allen folgenden Beispielen wird eine Taktfrequenz von 3,03 MHz entsprechend einer Zykluszeit von 0,330 µs zugrunde gelegt. Die Abfrage liegt am Ende der Schleife.

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
2880	3E		ORG	2880H
	FF		MVI	A,0FFH
2882	3D	PAUL	DCR	A
2883	C2		JNZ	PAUL
	82			
	28			
2886		MAX		

Berechnung der Ausführungszeit:

Schleife 15 X 255 = 3825 Zyklen
Laden von FF = 7 Zyklen
zusammen = 3832 Zyklen

Bei einer Zykluszeit von 0,330 µs liefert diese Schleife eine Verzögerung von 1,26 ms. Beginnt man mit dem Ausgangswert 00 und fragt wieder am Ende der Schleife ab, so lassen sich 256 Durchläufe erzielen.

Die folgenden Zählbefehle erhöhen bzw. vermindern den Inhalt eines 16-Bit-Registerpaares. Sie dienen zum Aufbau von Zählern in den Registerpaaren B,C und D,E und H,L. Der Stapelzeiger sollte nur zur Adressierung des Stapels verwendet werden. Die 16-Bit-Zählbefehle beeinflußen nicht das Bedingungsregister!

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
INX	B	code-eigen	03	1	5		Erhöhe Registerpaar B,C um 1
INX	D	code-eigen	13	1	5		Erhöhe Registerpaar D,E um 1
INX	H	code-eigen	23	1	5		Erhöhe Registerpaar H,L um 1
INX	S	code-eigen	33	1	5		Erhöhe Stapelzeiger um 1
DCX	B	code-eigen	0B	1	5		Vermindere Registerpaar B,C um 1
DCX	D	code-eigen	1B	1	5		Vermindere Registerpaar D,E um 1
DCX	H	code-eigen	2B	1	5		Vermindere Registerpaar H,L um 1
DCX	S	code-eigen	3B	1	5		Vermindere Stapelzeiger um 1

INX bedeutet INcrementiere X-Register, DCX bedeutet DeCrementiere X-Register.

Beispiel: Im Registerpaar D,E ist ein Abwärtszähler mit der größtmöglichen Zahl von Durchläufen (FFFF) aufzubauen. Für die Abfrage auf Null gibt es zwei Möglichkeiten:

a.) Ein Register des Paares wird in den Akkumulator gebracht und mit dem anderen Register durch das logische ODER (Befehl ORA) verknüpft. Da das Ergebnis der ODER-Funktion nur dann Null ist, wenn beide Operanden Null sind und der ORA-Befehl das Bedingungsregister beeinflußt, kann ein nachfolgender bedingter Sprungbefehl den Nulldurchgang des Registerpaares kontrollieren.

b.) Das Registerpaar wird mit einem Anfangswert geladen, die beiden Teile werden jedoch getrennt herabgezählt, da die 8-Bit-Zählbefehle das Bedingungsregister beeinflussen. Die beiden Teile des Registers dürfen nicht Null sein, da die Abfrage am Ende der Schleife liegt.

Lösung a

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	11	START	LXI	D,OFFFFH
	FF			
	FF			
2883	1B	DELI	DCX	D
2884	7A		MOV	A,D
2885	B3		ORA	E
2886	C2		JNZ	DELI
	83			
	28			
2889		MAX		

Lösung b

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	11	START	LXI	D,OFFFFH
	FF			
	FF			
2883	1D	DELI	DCR	E
2884	C2		JNZ	DELI
	83			
	28			
2887	15		DCR	D
2888	C2		JNZ	DELI
	83			
	28			
288B		MAX		

Ausführungszeit der Lösung a:

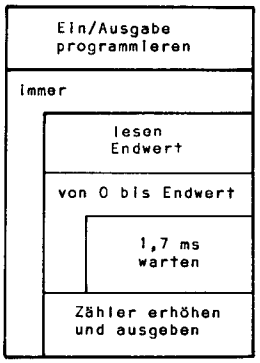
Schleife	24 X 65535	= 1572840 Zyklen
Laden mit FFFF		= 10 Zyklen
zusammen		= 1572850 Zyklen

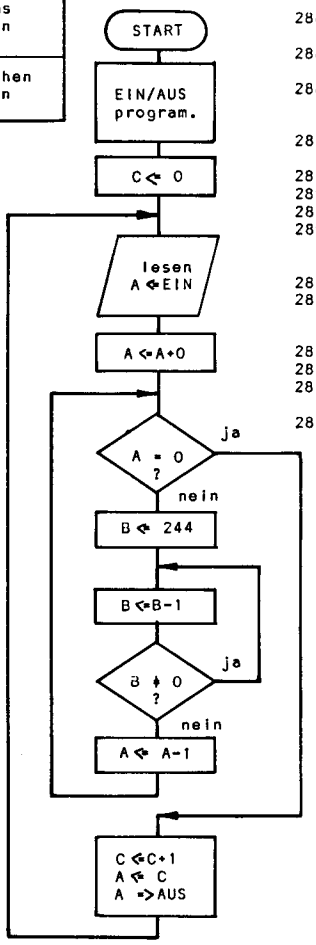
Bei einer Zykluszeit von 0,330 µs liefert diese Schleife eine Verzögerung von 519 ms. Beginnt man mit dem Ausgangswert 0000 und fragt wieder am Ende der Schleife ab, so lassen sich 65536 Durchläufe erzielen.
Längere Zeitverzögerungen lassen sich durch eine zweite äußere Schleife oder eine Verlängerung der bestehenden Schleife erreichen. Dazu kann der NOP-Befehl dienen, der eine Zeitverzögerung von vier Zyklen bringt, aber keine Operation ausführt. Er wird oft auch dazu benutzt, in einem Programm Platz für noch einzubauende Befehle frei zu halten oder die Lücken nicht mehr benötigter Befehle auszufüllen. Größere Lücken kann man auch einfach überspringen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
NOP		code-eigen	00	1	4		Keine Operation, Erhöhung des Befehlszähl.

NOP bedeutet No OPeration = keine Operation.

Das folgende Beispiel enthält zwei geschachtelte Zählschleifen. Eine innere Schleife im B-Register soll konstant um 1,7 ms verzögern, eine äußere Schleife soll im Akkumulator mit einem Ausgangswert laufen, der von einer Parallelschnittstelle Kanal 01 übernommen wird. Die Laufzeiten beider Schleifen multiplizieren sich. Nach ihrem Ablauf soll ein dritter Zähler, der im C-Register läuft, um 1 erhöht und anschließend über das Ausgaberegister 00 ausgegeben werden. Schließt man eine äußere Schaltung nach Bild 48 an, so kann man über Kippschalter Laufzeitfaktoren eingeben und an den Leuchtdioden den Zähler verfolgen.





Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	D3		OUT	02H
	02			
2884	3E		MVI	A,00H
	00			
2886	D3		OUT	03H
	03			
2888	0E		MVI	C,00H
	00			
288A	DB	SUSI	IN	01H
	01			
288C	C6		ADI	00H
	00			
288E	CA	PAUL	JZ	MAUS
	9D			
	28			
2891	06		MVI	B,0F4H
	F4			
2893	05	BUBI	DCR	B
2894	00		NOP	
2895	00		NOP	
2896	C2		JNZ	BUBI
	93			
	28			
2899	3D		DCR	A
289A	C3		JMP	PAUL
	8E			
	28			
289D	0C	MAUS	INR	C
289E	79		MOV	A,C
289F	D3		OUT	00H
	00			
28A1	C3		JMP	SUSI
	8A			
	28			
			END	

Berechnung der inneren Schleife:

```
Schleife     21 X 244   = 5124 Zyklen
Laden F4 = 244          =    7 Zyklen
zusammen                = 5131 Zyklen   = 1,693 ms
```

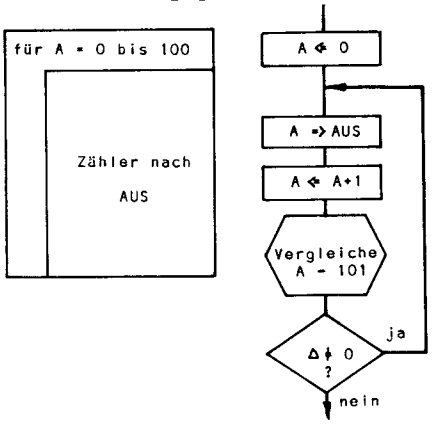
b. Aufwärtszähler mit Abfrage auf Endwert

Schleifen, die Adreßrechnungen in Registerpaaren durchführen, zählen meist von einem Anfangswert aufwärts bis zu einem Endwert. Da die Sprungbefehle nur auf Null abfragen, muß die Differenz zwischen dem laufenden Wert und dem Endwert gebildet werden. Die folgenden Vergleichsbefehle subtrahieren vom Inhalt des Akkumulators einen Vergleichswert und beeinflussen mit dem Ergebnis das Bedingungsregister, ohne jedoch die Differenz wie bei den Subtraktionsbefehlen in den Akkumulator zu übernehmen. Der laufende Zählerstand bleibt damit erhalten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ	H	P	C	
CMP	A	code-eigen	BF	1	4	xx	x	x	x	Vergleiche Akkumulator - Akkumulator
CMP	B	code-eigen	B8	1	4	xx	x	x	x	Vergleiche Akkumulator - B-Register
CMP	C	code-eigen	B9	1	4	xx	x	x	x	Vergleiche Akkumulator - C-Register
CMP	D	code-eigen	BA	1	4	xx	x	x	x	Vergleiche Akkumulator - D-Register
CMP	E	code-eigen	BB	1	4	xx	x	x	x	Vergleiche Akkumulator - E-Register
CMP	H	code-eigen	BC	1	4	xx	x	x	x	Vergleiche Akkumulator - H-Register
CMP	L	code-eigen	BD	1	4	xx	x	x	x	Vergleiche Akkumulator - L-Register
CPI	konst	unmittelbar	FE	2	7	xx	x	x	x	Vergleiche Akkumulator - 1-Byte-Konst.

CMP bedeutet CoMPare = vergleiche.

Beispiel: Im Akkumulator ist ein Aufwärtszähler von 0 bis 100 aufzubauen, dessen Zählerstand laufend zur Speicherstelle AUS zu bringen ist. Die Abfrage auf das Ende der Schleife übernimmt der Befehl "Springe bei ungleich Null". Da der Zähler nach Ausgabe des laufenden Wertes erhöht wird, vergleicht der Befehl CPI mit dem Endwert $100 + 1 = 101_{10} = 65_{16}$, damit der Endwert 100 noch ausgegeben wird.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	EQU	2000H
			ORG	2880H
2880	3E	START	MVI	A,00H
	00			
2882	32	SUSI	STA	AUS
	00			
	20			
2885	3C		INR	A
2886	FE		CPI	65H
	65			
2888	C2		JNZ	SUSI
	82			
	28			
288B		MAX		

Aufwärtszähler, die in den 8-Bit-Registern B,C,D,E,H oder L laufen, können nur mit Hilfe des Akkumulators auf einen Endwert verglichen werden.

Aufwärtszähler, die in den Registerpaaren B,C oder D,E oder H,L laufen und für die Registerpaar-Adressierung benötigt werden, kontrolliert man besser mit einem Abwärtszähler, der in einem anderen Register auf Null geprüft wird.

Übungen zum Abschnitt 4.4. **A**

1.Aufgabe

An den Mikrocomputer ist eine Schaltung nach Bild 48 angeschlossen. An dem A-Register der Parallelschnittstelle liegen acht Leuchtdioden, an die Klemme A0 kann zusätzlich ein Oszilloskop oder über einen Treiber ein Lautsprecher angeschlossen werden. Das Ausgaberegister ist laufend so um 1 zu erhöhen, daß am Ausgang A0 eine Rechteckfrequenz von 440 Hz (Kammerton A) auftritt. Periode 2,27 ms, Wartezeit für das Umschalten 1,136 ms.

2.Aufgabe

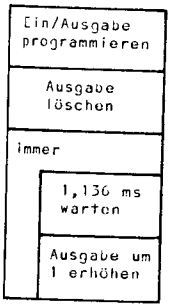
An den Mikrocomputer ist eine Schaltung nach Bild 48 angeschlossen, die als Würfel betrieben werden soll. An den Ausgängen A0 bis A2 sollen Dualzahlen im Bereich von 1 bis 6 erscheinen. Der Eingabeschalter B7 dient zur Steuerung. Ist der Schalter 0, so läuft der Zähler. Ist der Schalter 1, so hält der Zähler an. Da der Zähler sehr schnell im Vergleich zur Reaktionsfähigkeit des Menschen läuft, entscheidet der Zufall, bei welchem Zählerstand er gestoppt wird.

3.Aufgabe

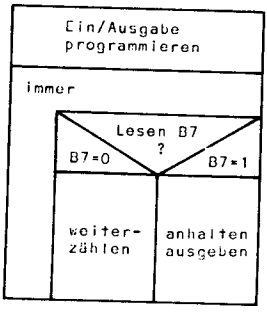
An den Mikrocomputer ist eine Schaltung nach Bild 48 angeschlossen. Vom Schalterfeld ist ein beliebiges Bitmuster zu lesen und an den Leuchtdioden mit einer Blinkfrequenz von 1 Hz (Halbperiode 500 ms) anzuzeigen.

Programmblockpläne

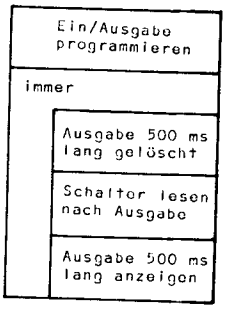
1.Aufgabe



2.Aufgabe



3.Aufgabe



4.5 Die Berechnung von Datenadressen über Indexregister

Bei den bisher behandelten Adressierungsarten stand die Datenadresse hinter dem Funktionscode im Befehl, sie wurde bei der Ausführung des Befehls in einen Zwischenspeicher gebracht, der dem Programmierer nicht zugänglich ist, und dann auf den Adreßbus gegeben. Die Adresse konnte nicht mehr verändert werden. Zum Ansprechen von 100 aufeinanderfolgenden Datenbytes waren damit 100 Befehle mit 100 verschiedenen Adressen erforderlich.

Bringt man dagegen die Datenadresse in ein Register, das der Programmierer über Befehle verändern kann, so sind damit Adreßrechnungen im Programm möglich. Bild 93 zeigt, wie die Datenadresse auch über ein Indexregister laufen kann, das mit dem Adreßbus verbunden ist.

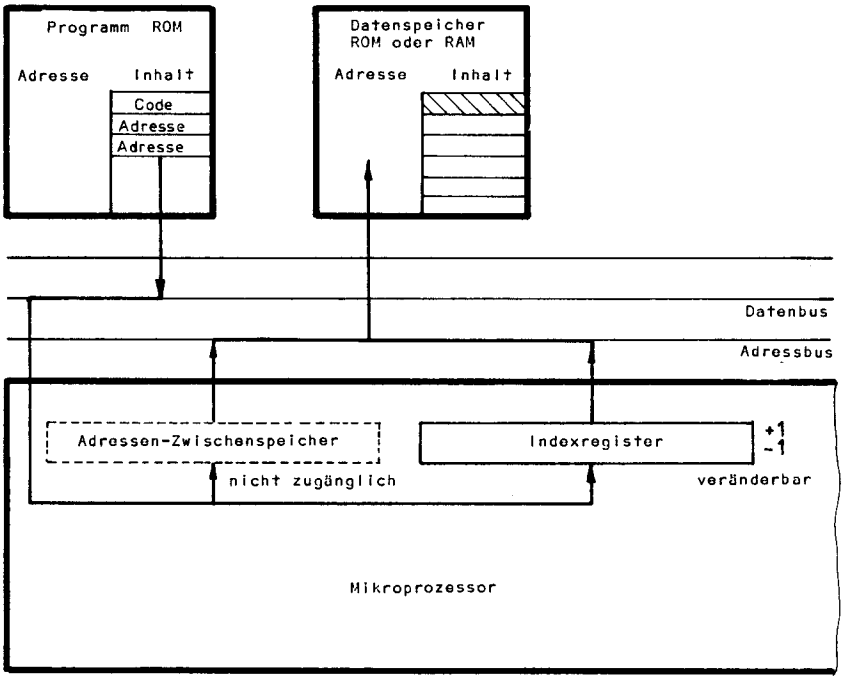


Bild 93: Indexregister für Datenadressen

Mit Hilfe des Indexregisters ist es nun möglich, eine Schleife aufzubauen, die die Adressen von 100 aufeinanderfolgenden Datenbytes berechnet.
Die drei Mikroprozessoren 6502(KIM-1), 6800(MOTOROLA) und 8080/8085(Intel) unterscheiden sich wesentlich in der Benutzung ihrer Indexregister.
Die beiden Indexregister des KIM-1 sind 8 bit lang. Zu der im Befehl stehenden Datenadresse (Basisadresse) wird der Inhalt des Indexregisters (Abstand) addiert.
Das Indexregister des 6800 ist 16 bit lang und nimmt eine vollständige Datenadresse (Basisadresse) auf. Alle Befehle, die mit

dem Indexregister Adressen ändern, enthalten im zweiten Byte des Befehls einen Wert, der zum Inhalt des Indexregisters addiert wird (Abstand).

Die Registerpaare des 8080/8085 sind 16 bit lang und enthalten eine Datenadresse (Basisadresse), die durch den Befehl nicht mehr verändert wird. Der Abstand ist also 0. Alle mit Registerpaar-Adressierung arbeitenden Befehle bestehen nur aus einem Byte, das neben dem Funktionscode die Bezeichnung des Registerpaares enthält.

4.5.1 Adreßrechnungen mit den Indexregistern des KIM-1 KIM

Der Mikroprozessor 6502 enthält zwei Indexregister X und Y zur Berechnung von Adressen. Der Inhalt eines Indexregisters besteht aus einer positiven Dualzahl im Bereich von 0 bis 255, die über einen Addierer zur absoluten (16 bit) oder zero-page (8 bit) Adresse des Befehls addiert wird. Die Summe ergibt die endgültige Datenadresse, die auf den Adreßbus gegeben wird.

Im folgenden Beispiel Bild 94 enthält ein Ladebefehl die absolute Adresse 1111, die mit dem Inhalt des X-Registers zu verändern (modifizieren) ist. Das Indexregister habe den Inhalt 5, die errechnete Datenadresse ist 1111 + 5 = 1116.

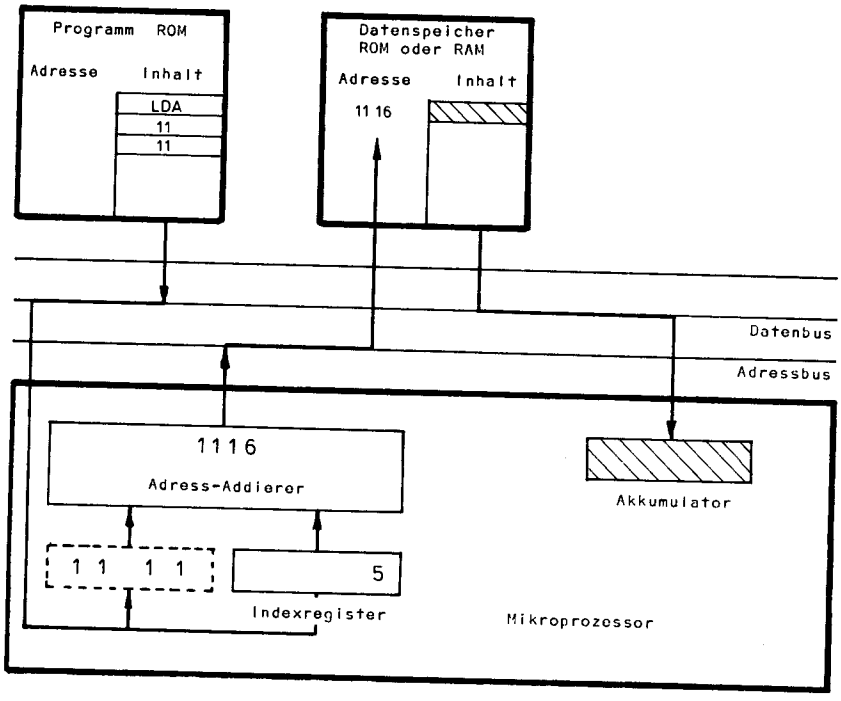


Bild 94: Adreßrechnung mit den Indexregistern des KIM-1

Bei der absoluten Adressierung (2 Bytes) wird zunächst das wertniedrigere Byte der Adresse aus dem Programmspeicher geholt und mit dem Inhalt des Indexregisters (1 Byte) addiert.
Das Ergebnis entscheidet über die Ausführungszeit des Befehls.

a.) Ist die Summe kleiner als 256, so tritt kein Übertrag auf das werthöhere Byte der Adresse auf, das nach seinem Eintreffen unverändert zur Adressierung der Datenspeicherstelle benutzt werden kann. Die Ausführung des Befehls dauert vier Zyklen bei allen Befehlen, die Daten aus dem Speicher holen.

Beispiel:	X-Register	05	
	alte Adresse	1111	Seite 11
	neue Adresse	1116	Seite 11

b.) Ist die Summe größer als 256, so tritt ein Übertrag auf das werthöhere Byte der Adresse auf, das nach seinem Eintreffen nun ebenfalls über den Adreßaddierer laufen muß, bevor es zur Adressierung verwendet werden kann. Die Ausführung des Befehls dauert fünf Zyklen.

Beispiel:	X-Register	FA	
	alte Adresse	1111	Seite 11
	neue Adresse	120B	Seite 12

Bei allen Befehlen, die Speicherinhalte verändern (z.B.speichern) wird immer eine mögliche zweite Addition berücksichtigt. Liefert die Adreßrechnung einen Übertrag zum werthöheren Byte der Adresse, so spricht man von einer Seitenüberschreitung.

Bei der zero-page Adressierung besteht die ursprüngliche Datenadresse des Befehls nur aus einem Byte, die adressierte Speicherstelle steht in Seite Null. Bei der Addition dieses Bytes zum Inhalt des Indexregisters können zwei Fälle auftreten:

a.) Die Summe ist kleiner als 256, es tritt kein Übertrag auf. Damit liegt die neue Adresse ebenfalls in Seite Null.

Beispiel:	X-Register	24	
	alte Adresse	13	Seite Null
	neue Adresse	37	Seite Null

b.) Die Summe ist größer als 255, es tritt bei der Addition ein Übertrag auf, der jedoch nicht berücksichtigt wird, weil bei jeder zero-page Adressierung das werthöhere Byte der Adresse durch den Prozessor automatisch Null gesetzt wird.

Beispiel:	X-Register	F4	
	alte Adresse	13	Seite Null
	neue Adresse	07	Seite Null
			richtig wäre 107 Seite 1

Der Mikroprozessor sendet die verstümmelte Adresse 07 anstelle des richtigen Wertes 107 aus. In den meisten Fällen dürfte es sich hierbei um einen Fehler handeln, wenn der Programmierer einen Speicherbereich fortlaufend adressieren will und annimmt, daß auf das Ende der Seite 0 der Anfang der Seite 1 folgt.

Die folgende Tabelle zeigt die bisher behandelten Befehle mit den beiden neuen Adressierungsarten absolut-indiziert und zero-page-indiziert für die beiden Indexregister X und Y.

Ladebefehle indiziert:

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
LDA	Adres,X	absol. ind.	BD	3	4/5	X				X		Lade Akkumulator (absolut modifiziert mit X)
LDA	Adres,Y	absol. ind.	B9	3	4/5	X				X		Lade Akkumulator (absolut modifiziert mit Y)
LDAZ	Adres,X	zer-pag ind.	B5	2	4	X				X		Lade Akkumulator (zero-page modifiziert mit X)
LDX	Adres,Y	absol. ind.	BE	3	4/5	X				X		Lade X-Register (absolut modifiziert mit Y)
LDXZ	Adres,Y	zer-pag ind.	B6	2	4	X				X		Lade X-Register (zero-page modifiziert mit Y)
LDY	Adres,X	absol. ind.	BC	3	4/5	X				X		Lade Y-Register (absolut modifiziert mit X)
LDYZ	Adres,X	zer-pag ind.	B4	2	4	X				X		Lade Y-Register (zero-page modifiziert mit X)

Speicherbefehle indiziert:

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
STA	Adres,X	absol. ind.	9D	3	5							Speichere Akku. (absolut modifiziert mit X)
STA	Adres,Y	absol. ind.	99	3	5							Speichere Akku. (absolut modifiziert mit Y)
STAZ	Adres,X	zer-pag ind.	95	2	4							Speichere Akku. (zero-page modifiziert mit X)
STXZ	Adres,Y	zer-pag ind.	96	2	4							Speichere X-Reg. (zero-page modifiziert mit Y)
STYZ	Adres,X	zer-pag ind.	94	2	4							Speichere Y-Reg. (zero-page modifiziert mit X)

Zählbefehle indiziert:

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
INC	Adres,X	absol.,ind.	FE	3	7	X				X		Erhöhe Speicher um 1 absolut mod. mit X
INCZ	Adres,X	zer-pag,ind.	F6	2	6	X				X		Erhöhe Speicher um 1 zero-page mod. mit X
DEC	Adres,X	absol.,ind.	DE	3	7	X				X		Vermin. Speicher um 1 absolut mod. mit X
DECZ	Adres,X	zer-pag,ind.	D6	2	6	X				X		Vermin. Speicher um 1 zero-page mod. mit X

Vergleichsbefehle indiziert:

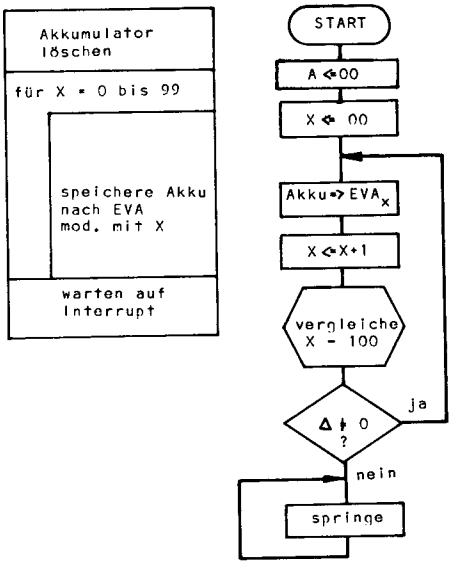
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
CMP	Adres,X	absol.,ind.	DD	3	4/5	X				X	X	Vergleiche Akku mit Speicher absol. mod. X
CMP	Adres,Y	absol.,ind.	D9	3	4/5	X				X	X	Vergleiche Akku mit Speicher absol. mod. Y
CMPZ	Adres,X	zer-pag,ind.	D5	2	4	X				X	X	Vergleiche Akku mit Speicher zero-p mod. X

Das X bzw. Y hinter der Adresse bedeutet, daß die Adresse noch mit dem Inhalt eines der beiden Indexregister verändert wird.

Stehen in der Spalte Zyklen zwei durch einen Schrägstrich getrennte Werte, so gilt der linke Wert für den Fall, daß bei der Adreßrechnung keine Seitenüberschreitung auftritt, der rechte für den Fall der Seitenüberschreitung.

Läßt man in einem der beiden Indexregister einen Aufwärtszähler laufen, so kann man durch indizierte Adressierung einen Datenbereich von einer Anfangsadresse bis zu einer Endadresse erreichen. Der Endwert wird meist mit einem Vergleichsbefehl kontrolliert.

Beispiel:
Es ist ein Bereich von 100 Bytes mit der Anfangsadresse EVA mit binären Nullen zu füllen. Die folgende Lösung benutzt eine Indexschleife, in der die Datenadresse des Befehls mit Hilfe des X-Registers laufend von 0 bis 99 erhöht wird.



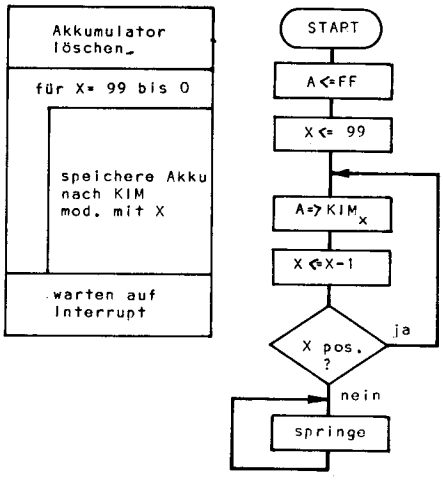
Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0000	
0000		EVA	*=*+100	
			*=$0300	
0300	A9	START	LDA	#$00
	00			
0302	A2		LDX	#$00
	00			
0304	9D	ADAM	STA	EVA,X
	00			
	00			
0307	E8		INX	
0308	E0		CPX	#$64
	64			
030A	D0		BNE	ADAM
	F8			
030C	4C	MAUS	JMP	MAUS
	0C			
	03			
			.END	

Der Befehl STA EVA,X erzeugt nacheinander die Adressen

EVA+00
EVA+01
EVA+02
⋮
EVA+97
EVA+98
EVA+99

Bei vielen Anwendungen ist es ohne Bedeutung, in welcher Reihenfolge die Bytes eines Bereiches angesprochen werden. Läßt man einen Abwärtszähler bis Null laufen, so vereinfacht sich die Kontrolle des Endwertes.

Beispiel: Es ist ein Bereich von 100 Bytes mit der Anfangsadresse KIM mit binären Einsen zu füllen. Der Bereich liegt in Seite Null (zero-page Adressierung). Die folgende Lösung benutzt einen Abwärtszähler im X-Register. Die Abfrage liegt am Ende der Schleife. Sie läuft solange, bis der Zähler negativ wird, damit der Endwert Null noch für die Indizierung zur Verfügung steht.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0000	
0000		KIM	*=*+100	
			*=$0300	
0300	A9	START	LDA	#$FF
	FF			
0302	A2		LDX	#$63
	63			
0304	95	OTTO	STAZ	KIM,X
	00			
0306	CA		DEX	
0307	10		BPL	OTTO
	FB			
0309	4C	MAUS	JMP	MAUS
	09			
	03			
			.END	

Der Befehl STAZ KIM,X erzeugt nacheinander die Adressen

KIM+99
KIM+98
KIM+97
⋮
KIM+02
KIM+01
KIM+00

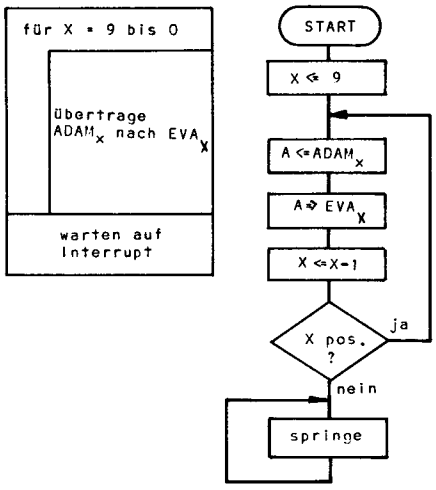
Ohne die Warteschleife besteht das Programm aus neun Bytes, es wird in 4 + 100X9 = 904 Zyklen ausgeführt. Ein entsprechendes Programm mit einem Lade- und 100 Speicherbefehlen besteht aus 2 + 100X2 = 202 Bytes und benötigt 2 + 100X3 = 302 Zyklen für die Ausführung. Die Indexschleife verkürzt also die Zahl der Befehle, verlängert jedoch die Laufzeit!

Da der Inhalt des Indexregisters zu der im Befehl enthaltenen Adresse addiert wird, kann ein Indexregister die Adressen mehrerer Befehle modifizieren.

Beispiel: Ein Bereich aus zehn Bytes mit der Anfangsadresse ADAM ist in einen gleichgroßen Bereich mit der Anfangsadresse EVA zu transportieren. Da die Reihenfolge, in der die Daten übertragen werden, keine Rolle spielt, enthält die Indexschleife im X-Register einen Abwärtszähler. Der Akkumulator dient als Zwischenspeicher.

Das X-Register berechnet folgende Adressen:

ADAM+9 ⟶ EVA+9
ADAM+8 ⟶ EVA+8
ADAM+7 ⟶ EVA+7
ADAM+6 ⟶ EVA+6
ADAM+5 ⟶ EVA+5
ADAM+4 ⟶ EVA+4
ADAM+3 ⟶ EVA+3
ADAM+2 ⟶ EVA+2
ADAM+1 ⟶ EVA+1
ADAM+0 ⟶ EVA+0

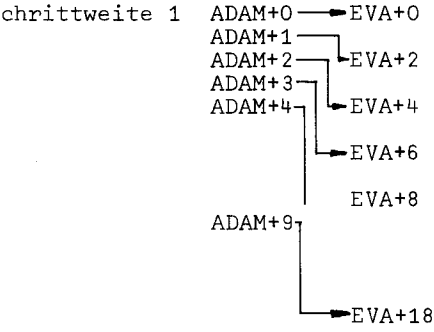


Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			*=$0000	
0000		EVA	*=*+10	
000A		ADAM	*=*+10	
			*=$0300	
0300	A2	START	LDX	#$09
	09			
0302	B5	OTTO	LDAZ	ADAM,X
	0A			
0304	95		STAZ	EVA,X
	00			
0306	CA		DEX	
0307	10		BPL	OTTO
	F9			
0309	4C	MAUS	JMP	MAUS
	09			
	03			
			.END	

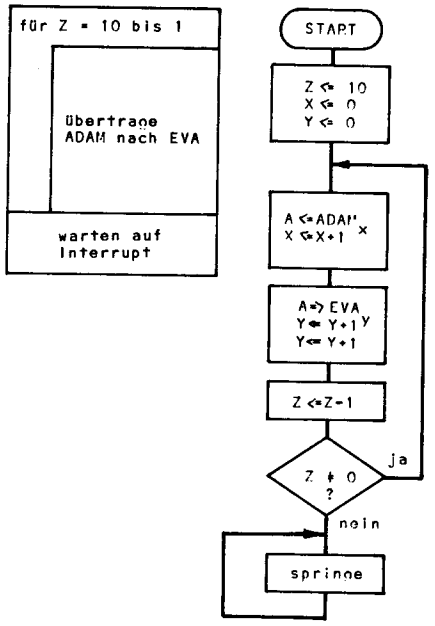
Da beide Indexregister nur acht bit lang sind, lassen sich in einer Indexschleife nur Bereiche von maximal 256 byte Länge adressieren; sind größere Bereiche anzusprechen, so sind entweder mehrere Schleifen zu verwenden oder es ist die indirekte Adressierung zu Hilfe zu nehmen.

Die bisher verwendeten Indexschleifen arbeiteten mit der Schrittweite 1. Besonders bei der Arbeit mit Tabellen kann es vorkommen, daß nur jedes zweite oder dritte Byte anzusprechen ist. Dazu kann man zwei oder drei Zählbefehle verwenden. Erst bei größeren Schrittweiten kann es zweckmäßig sein, die Adresse im Akkumulator über einen Additionsbefehl berechnen zu lassen und in ein Indexregister zu übertragen.

Beispiel: Das folgende Programm überträgt 10 hintereinanderliegende Bytes aus dem Bereich ADAM in jedes zweite Byte von EVA.



Das X-Register modifiziert die Adresse von ADAM mit der Schrittweite 1; das Y-Register modifiziert die Adresse von EVA mit der Schrittweite 2; eine Speicherstelle Z zählt die Schleifendurchläufe und der Akkumulator dient als Zwischenspeicher. Die Speicheradressen liegen zwar in Seite Null, es wird jedoch die absolute Adressierung verwendet.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			*=$0000	
0000		ADAM	*=*+10	
000A		EVA	*=*+20	
001E		Z	*=*+1	
			*=$0300	
0300	A9	START	LDA	#$0A
	0A			
0302	8D		STA	Z
	1E			
	00			
0305	A2		LDX	#$00
	00			
0307	A0		LDY	#$00
	00			
0309	BD	OTTO	LDA	ADAM,X
	00			
	00			
030C	E8		INX	
030D	99		STA	EVA,Y
	0A			
	00			
0310	C8		INY	
0311	C8		INY	
0312	CE		DEC	Z
	1E			
	00			
0315	D0		BNE	OTTO
	F2			
0317	4C	MAUS	JMP	MAUS
	17			
	03			
			.END	

Im Normalfall enthält der Befehl die Datenadresse (Basisadresse) und ein Indexregister einen Abstand, der zur Datenadresse addiert wird. Liegt die Speicherstelle in Seite Null, so genügt eine Adresse von acht bit Länge, die in einem Indexregister untergebracht werden kann. Der Adreßteil des Befehls enthält dann einen konstanten Abstand.

4.5.2 Adreßrechnungen mit dem Indexregister des 6800 **6800**

Der Mikroprozessor 6800 enthält ein Indexregister, das eine Adresse von 16 bit Länge aufnimmt. Dieses Register kann, wie bereits gezeigt, durch Befehle geladen, erhöht, vermindert, verglichen und in den Datenspeicher gebracht werden. Die bisher verwendeten Adressierungsarten zero-page (direct) und absolut (extended) enthielten im Befehl neben dem Funktionscode eine Datenadresse. Bei der nun zu behandelnden indizierten Adressierung (indexed) steht die Datenadresse im Indexregister; der Befehl enthält neben dem Funktionscode eine positive Dualzahl von einem byte Länge (offset), die zum Inhalt des Indexregisters dazuaddiert wird. Die Summe bildet die endgültige Adresse. Der Inhalt des Indexregisters mit der Basisadresse ist veränderbar, das zweite Byte des Befehls mit dem Abstand zur Basisadresse ist konstant.
Im folgenden Beispiel Bild 95 enthält das Indexregister die Basisadresse 1111, ein Ladebefehl mit dem Abstand 05 soll ein Byte aus dem Datenspeicher in den Akkumulator bringen. Die Summe 1111 + 05 = 1116 ist die errechnete Datenadresse.
Läßt man nun diesen Ladebefehl mit indizierter Adressierung in einer Schleife laufen, die den Inhalt des Indexregisters laufend um 1 erhöht, so wäre beim nächsten Durchlauf der neue Inhalt des Indexregisters 1112; die neue Datenadresse ist 1112 + 05 = 1117.

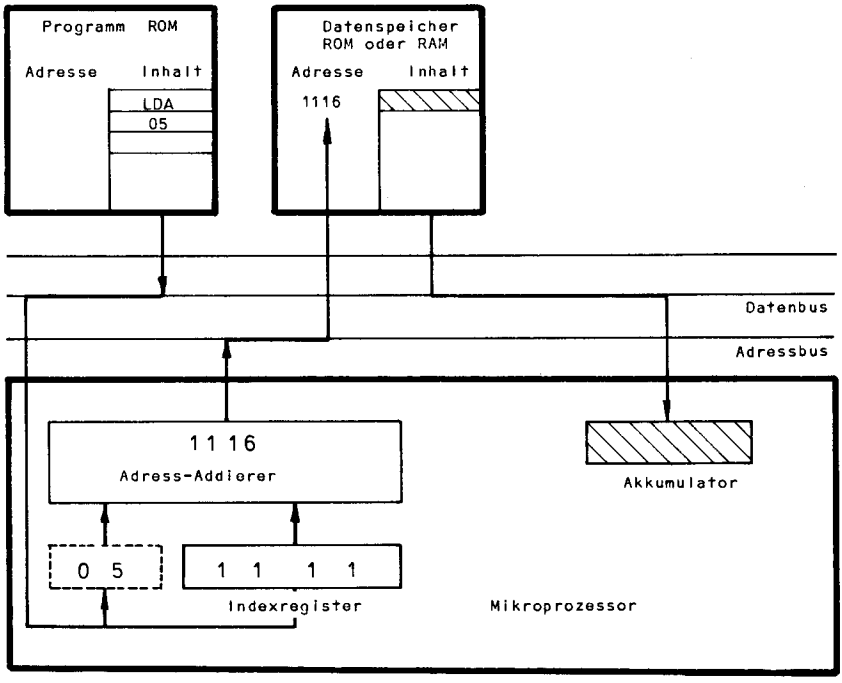


Bild 95: Adreßrechnung mit dem Indexregister des 6800

Die indizierte Adressierung läßt sich auf die folgenden bisher behandelten Befehle anwenden:

Ladebefehle indiziert

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
LDA A	konst,X	indiziert	A6	2	5	⇕⇕⇕0⇕	Lade Akkumulator A (indiziert adressiert)
LDA B	konst,X	indiziert	E6	2	5	⇕⇕⇕0⇕	Lade Akkumulator B (indiziert adressiert)
LDX	konst,X	indiziert	EE	2	6	⇕⇕⇕0⇕	Lade X-Register (indiziert adressiert)
LDS	konst,X	indiziert	AE	2	6	⇕⇕⇕0⇕	Lade Stapelzeiger (indiziert adressiert)

Speicherbefehle indiziert

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
STA A	konst,X	indiziert	A7	2	6	⇕⇕⇕0⇕	Speichere Akku A (indiziert adressiert)
STA B	konst,X	indiziert	E7	2	6	⇕⇕⇕0⇕	Speichere Akku B (indiziert adressiert)
STX	konst,X	indiziert	EF	2	7	⇕⇕⇕0⇕	Speichere X-Register (indiziert adressiert)
STS	konst,X	indiziert	AF	2	7	⇕⇕⇕0⇕	Speichere Stapelzeiger (indiziert addr.)
CLR	konst,X	indiziert	6F	2	7	0100	Lösche Speicherstelle (indiziert adressiert)

Zähl- und Vergleichsbefehle indiziert

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
INC	konst,X	indiziert	6C	2	7	↕ ↕	Erhöhe Speicherstelle um 1 (indiziert)
DEC	konst,X	indiziert	6A	2	7	↕ ↕	Vermindere Speicherstelle um 1 (indiziert)
CMP A	konst,X	indiziert	A1	2	5	↕ ↕ ↕ ↕	Vergleiche Akku A - Speicher (indiziert)
CMP B	konst,X	indiziert	E1	2	5	↕ ↕ ↕ ↕	Vergleiche Akku B - Speicher (indiziert)
CPX	konst,X	indiziert	AC	2	6	↕ ↕	Vergleiche X-Reg. - Speicher (indiziert)

Das Zeichen X hinter den Konstanten bedeutet, daß zu diesem konstanten Abstand noch der Inhalt des Indexregisters addiert wird; die Summe ist die endgültige Datenadresse.

Alle Befehle in der indizierten Adressierung erhalten in der Assemblerschreibweise im Operandenteil eine zweistellige Hexadezimalzahl, die in das zweite Byte des Befehls gesetzt wird.

Beispiel: STA A 05,X

Das Indexregister wurde bisher nur mit Daten, z.B. Zählern geladen. Die Daten konnten in Speicherstellen oder als Direktoperanden im zweiten und dritten Byte des Befehls stehen.

1.Lösung: Speicherstelle EVA enthält den Wert FFFF

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000            EVA    FDB      $FFFF

                       ORG      $0180
0180      FE           LDX      EVA
          00
          00
```

Lade das Indexregister mit dem Inhalt von EVA.

2.Lösung: Direktoperand FFFF im Befehl

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0180
0180      CE           LDX      #$FFFF
          FF
          FF
```

Lade das Indexregister mit der Konstanten FFFF.

Bei der indizierten Adressierung muß vorher das Indexregister mit Adressen geladen werden. Diese Adressen können in Speicherstellen oder auch als Direktoperanden im Befehl stehen. In den folgenden Beispielen ist ein Bereich EVA mit 100 Speicherstellen zu reservieren. Die Anfangsadresse EVA ist in das Indexregister zu laden.

1.Lösung: Die Adresse EVA steht in der Speicherstelle ADAM

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000            EVA    RMB      100
0064    00      ADAM   FDB      EVA
        00

                       ORG      $0180
0180    FE             LDX      ADAM
        00
        64
```

Lade das Indexregister mit dem Inhalt der Speicherstelle ADAM, die die Adresse von EVA enthält.

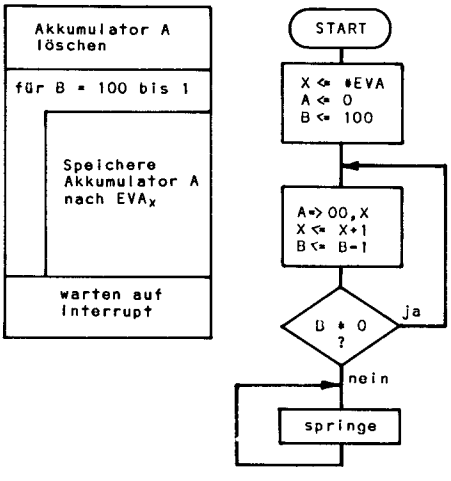
2.Lösung: Die Adresse EVA steht als Direktoperand im Befehl.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000            EVA    RMB      100

                       ORG      $0180
0180    CE             LDX      #EVA
        00
        00
```

Lade das Indexregister mit der Adresse der Speicherstelle EVA.

Lädt man das Indexregister mit der Anfangsadresse eines Bereiches und erhöht es anschließend in einer Schleife, so kann man alle Bytes des Bereiches nacheinander adressieren.

Beispiel: Es ist ein Bereich von 100 Bytes von der Anfangsadresse EVA ab mit binären Nullen zu füllen. Das Indexregister soll aufwärts gezählt werden, der Abstand zur Anfangsadresse ist 00. Der Akkumulator wird vor der Schleife gelöscht und laufend nach EVA indiziert übertragen; es wäre jedoch auch der Löschbefehl CLR möglich. Im Akkumulator B läuft ein Zähler, der die Schleifendurchläufe kontrolliert.




```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000            EVA    RMB      100

                       ORG      $0180
0180    CE     START   LDX      #EVA
        00
        00
0183    4F             CLR A
0184    C6             LDA B    #$64
        64
0186    A7     MAX     STA A    00,X
        00
0188    08             INX
0189    5A             DEC B
018A    26             BNE      MAX
        FA
018C    7E     MAUS    JMP      MAUS
        01
        8C
                       END
```

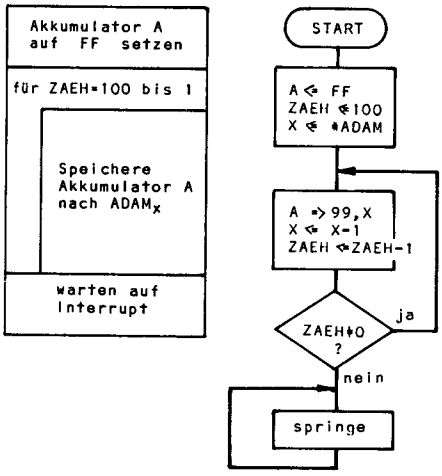
Die folgende Tabelle zeigt die Adreßrechnung dezimal:

Zähler Akku B	X-Register	+ Abstand	= Adresse	symbolisch
100	0000	00	0000	EVA+0
99	0001	00	0001	EVA+1
98	0002	00	0002	EVA+2
⋮	⋮	⋮	⋮	⋮
3	0097	00	0097	EVA+97
2	0098	00	0098	EVA+98
1	0099	00	0099	EVA+99

Das Programm besteht ohne Warteschleife aus 12 Bytes, es wird in 7 + 100X16 = 1607 Zyklen ausgeführt. Ein entsprechendes Programm mit einem Lade- und 100 Speicherbefehlen besteht aus 2 + 100X2 = 202 Bytes (zero-page Adressierung) und benötigt 2 + 100X4 = 402 Zyklen. Die Indexschleife verringert die Zahl der Befehle, verlängert jedoch die Laufzeit.

Bei vielen Anwendungen können die Bytes eines Bereiches in beliebiger Reihenfolge angesprochen werden. Ist die Zahl der Elemente kleiner als 256, so kann mit Hilfe des Abstandes im zweiten Byte des Befehls das Indexregister von der Endadresse des Bereiches herabgezählt werden. Sind mehr als 256 Elemente vorhanden, so muß das Indexregister mit der Endadresse geladen werden.

Beispiel: Ein Bereich von 100 Bytes mit der Anfangsadresse ADAM sei mit binären Einsen zu füllen. Die folgende Lösung zählt das Indexregister abwärts und kontrolliert das Ende der Schleife mit einem Zähler, der in einer Speicherstelle abwärts läuft.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	$0000
0000		ADAM	RMB	100
0064		ZAEH	RMB	1
			ORG	$0180
0180	86 FF	START	LDA A	#$FF
0182	C6 64		LDA B	#$64
0184	F7 00 64		STA B	ZAEH
0187	CE 00 00		LDX	#ADAM
018A	A7 63	OTTO	STA A	$63,X
018C	09		DEX	
018D	7A 00 64		DEC	ZAEH
0190	26 F8		BNE	OTTO
0192	7E 01 92	MAUS	JMP	MAUS
			END	

Die folgende Tabelle zeigt dezimal die Adreßrechnung:

Zähler	Indexregister	+ Abstand	= Adresse	symbolisch
100	0000	99	0099	ADAM+99
99	-1	99	0098	ADAM+98
97	-2	99	0097	ADAM+97
⋮	⋮	⋮	⋮	⋮
3	-97	99	0002	ADAM+2
2	-98	99	0001	ADAM+1
1	-99	99	0000	ADAM+0

Durch mehrere aufeinanderfolgende Zählbefehle lassen sich längere konstante Schrittweiten erreichen.

Beispiel: Indexschleife mit der Schrittweite drei.

```
  Hedadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000    0C      ZAE    FCB      $0C

                       ORG      $0180
0180    A7      OTTO   STA A    00,X
        00
0182    08             INX
0183    08             INX
0184    08             INX
0185    7A             DEC      ZAE
        00
        00
0188    26             BNE      OTTO
        F6
018A
```

Ist die Schrittweite variabel, so kann ein Zähler mitlaufen, der die Anzahl der Zählbefehle steuert.

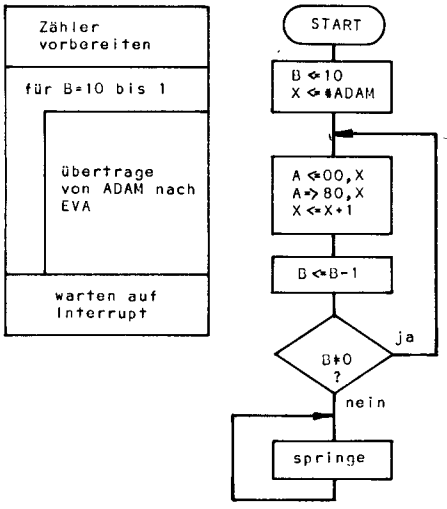
Beispiel: Die Speicherstelle MAX enthält eine positive Schrittweite.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000    64      ZAE    FCB      $64
0001    03      MAX    FCB      $03

                       ORG      $0180
0180    A7      OTTO   STA A    00,X
        00
0182    F6             LDA B    MAX
        00
        01
0185    08      SUSI   INX
0186    5A             DEC B
0187    26             BNE      SUSI
        FC
0189    7A             DEC      ZAE
        00
        00
018C    26             BNE      OTTO
        F2
018E
```

Der Mikroprozessor 6800 enthält nur ein Indexregister, bei der Übertragung von Daten zwischen zwei Bereichen müssen jedoch zwei Adressen laufend verändert werden. In dem einfachen Fall,

daß die beiden Anfangsadressen nicht mehr als 255 Bytes auseinanderliegen und daß die Schrittweiten beider Bereiche gleich sind, kann man mit Hilfe des Abstandes mit einer gemeinsamen Basisadresse im Indexregister arbeiten.

Beispiel: Ein Speicherbereich ADAM (Anfangsadresse 0000) mit 10 Bytes soll in einen Speicherbereich EVA (Anfangsadresse 0080) gebracht werden. Die gemeinsame Basisadresse ist 0000, der Anfang von ADAM hat den Abstand 00, der Anfang von EVA den Abstand 80. Der Akkumulator B zählt die Schleifendurchläufe.



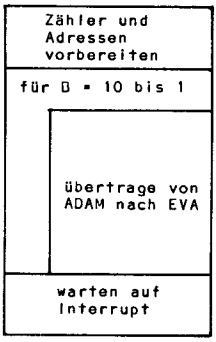
Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	$0000
0000		ADAM	RMB	10
			ORG	$0080
0080		EVA	RMB	10
			ORG	$0180
0180	C6	START	LDA B	#$0A
	0A			
0182	CE		LDX	#ADAM
	00			
	00			
0185	A6	SUSI	LDA A	00,X
	00			
0187	A7		STA A	$80,X
	80			
0189	08		INX	
018A	5A		DEC B	
018B	26		BNE	SUSI
	F8			
018D	7E	KATZ	JMP	KATZ
	01			
	8D			
			END	

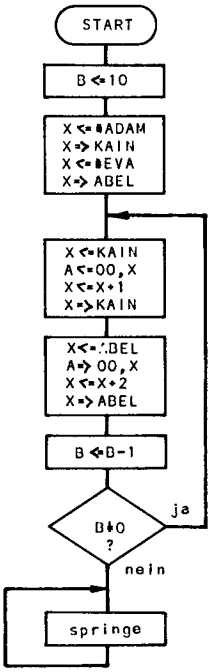
Die folgende Tabelle zeigt die Adreßrechnungen dezimal:

Zähler	X-Reg.	+ Abst.	= Adresse	symb.	X-Reg.	+ Abst.	=Adresse	sym.
10	0000	00	0000	ADAM+0	0000	80	0080	EVA+0
9	0001	00	0001	ADAM+1	0001	81	0081	EVA+1
8	0002	00	0002	ADAM+2	0002	82	0082	EVA+2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2	0008	00	0008	ADAM+8	0008	88	0088	EVA+8
1	0009	00	0009	ADAM+9	0009	89	0089	EVA+9

Liegen die Anfangsadressen beider Bereiche mehr als 255 Bytes auseinander oder haben die beiden Indexzähler verschiedene Schrittweiten, so muß bei der Adressierung zweier Bereiche das Indexregister abwechselnd die beiden laufenden Adressen enthalten.

Beispiel: Das folgende Programm überträgt zehn hintereinanderliegende Bytes aus einem Bereich ADAM (Anfangsadresse 0000) in jedes zweite Byte des Bereiches EVA (Anfangsadresse 0100). Die Speicherstellen KAIN und ABEL dienen als Zwischenspeicher für die Indexzähler. Der Akkumulator B zählt die Schleifendurchläufe.





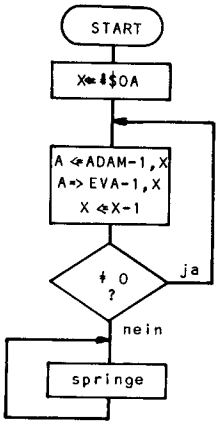
Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	$0000
0000		ADAM	RMB	10
000A		KAIN	RMB	1
			ORG	$0100
0100		EVA	RMB	20
0114		ABEL	RMB	1
			ORG	$0180
0180	C6	START	LDA B	#$0A
	0A			
0182	CE		LDX	#ADAM
	00			
	00			
0185	FF		STX	KAIN
	00			
	0A			
0188	CE		LDX	#EVA
	01			
	00			
018B	FF		STX	ABEL
	01			
	14			
018E	FE	OTTO	LDX	KAIN
	00			
	0A			
0191	A6		LDA A	00,X
	00			
0193	08		INX	
0194	FF		STX	KAIN
	00			
	0A			
0197	FE		LDX	ABEL
	01			
	14			
019A	A7		STA A	00,X
	00			
019C	08		INX	
019D	08		INX	
019E	FF		STX	ABEL
	01			
	14			
01A1	5A		DEC B	
01A2	26		BNE	OTTO
	EA			
01A4	20	KATZ	BRA	KATZ
	FE			
			END	

Die folgende Tabelle zeigt die Adreßrechnungen:

Zähler Akku B	Zähler KAIN X-Reg.	+ Abst.	=Adresse	symb.	Zähler ABEL X-Reg.	+Abst.	= Adresse	symb.
10	0000	00	0000	ADAM+0	0100	00	0100	EVA+0
9	0001	00	0001	ADAM+1	0102	00	0102	EVA+2
8	0002	00	0002	ADAM+2	0104	00	0104	EVA+4
7	0003	00	0003	ADAM+3	0106	00	0106	EVA+6
6	0004	00	0004	ADAM+4	0108	00	0108	EVA+8
5	0005	00	0005	ADAM+5	010A	00	010A	EVA+10
4	0006	00	0006	ADAM+6	010C	00	010C	EVA+12
3	0007	00	0007	ADAM+7	010E	00	010E	EVA+14
2	0008	00	0008	ADAM+8	0110	00	0110	EVA+16
1	0009	00	0009	ADAM+9	0112	00	0112	EVA+18

In den bisherigen Beispielen enthielt das X-Register eine volle 16-Bit-Datenadresse (extended) und das 2. Byte des Befehls einen konstanten Abstand. Für den Fall, daß sich die Datenadresse in Seite Null (zero-page, directe Adresse) befindet, kann sie als

8-Bit-Adresse im 2. Byte des Befehls stehen, das Indexregister nimmt dann einen Abstand auf, der über Zählbefehle veränderbar ist und gleichzeitig die Schleifendurchläufe kontrolliert.
Beispiel: Die Bereiche ADAM und EVA bestehen aus je 10 Speicherstellen und befinden sich in Seite 0. Der Bereich ADAM ist nach EVA zu übertragen. Das 2. Byte der Lade- und Speicherbefehle enthält jeweils die Anfangsadresse, das Indexregister enthält den veränderlichen Abstand und dient gleichzeitig zum Zählen der Schleifendurchläufe.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	$0010
0010		ADAM	RMB	10
001A		EVA	RMB	10
			ORG	$0180
0180	CE 00 0A	START	LDX	#$000A
0183	A6 0F	SUSI	LDA A	ADAM-1,X
0185	A7 19		STA A	EVA-1,X
0187	09		DEX	
0188	26 F9		BNE	SUSI
018A	20 FE	OTTO	BRA	OTTO
			END	

4.5.3 Die Registerpaar-Adressierung des 8080/8085 8080

Der Mikroprozessor 8080 bzw. 8085 enthält drei 16-Bit-Registerpaare, die als Indexregister verwendet werden können.
Bei der Registerpaar-Adressierung ist der Name des verwendeten Registerpaares Bestandteil des Funktionscodes. Die entsprechenden Befehle sind 1-Byte-Befehle ohne Adreßteil; die Adresse des anzusprechenden Datenbytes steht in einem der drei Registerpaare und wird durch den Befehl nicht mehr verändert. Dafür gibt es besondere Befehle, die den Inhalt eines Registerpaares laden, erhöhen und vermindern.

Im folgenden Beispiel Bild 96 enthält das Registerpaar H,L die Datenadresse 1116, ein Ladebefehl lädt den Akkumulator mit der Speicherstelle, deren Adresse in diesem Registerpaar steht, also mit dem Inhalt der Speicherstelle 1116.

Bei der Ausführung des Befehls gelangt das einzige Byte des Befehls mit dem Funktionscode in das Befehlsregister, als Adresse wird der Inhalt des Registerpaares H,L auf den Adreßbus gegeben. Liegt nun dieser Lade-Befehl mit Registerpaar-Adressierung in einer Schleife, die den Inhalt des Registerpaares laufend um 1 erhöht, so wäre beim nächsten Durchlauf die Datenadresse 1117.

Mit Hilfe des LXI-Befehls ist es möglich, ein Registerpaar mit einer Konstanten zu laden, die als Direktoperand im 2. und 3. Byte des Befehls steht. Dies kann eine Zahlenkonstante oder eine Adreßkonstante sein.

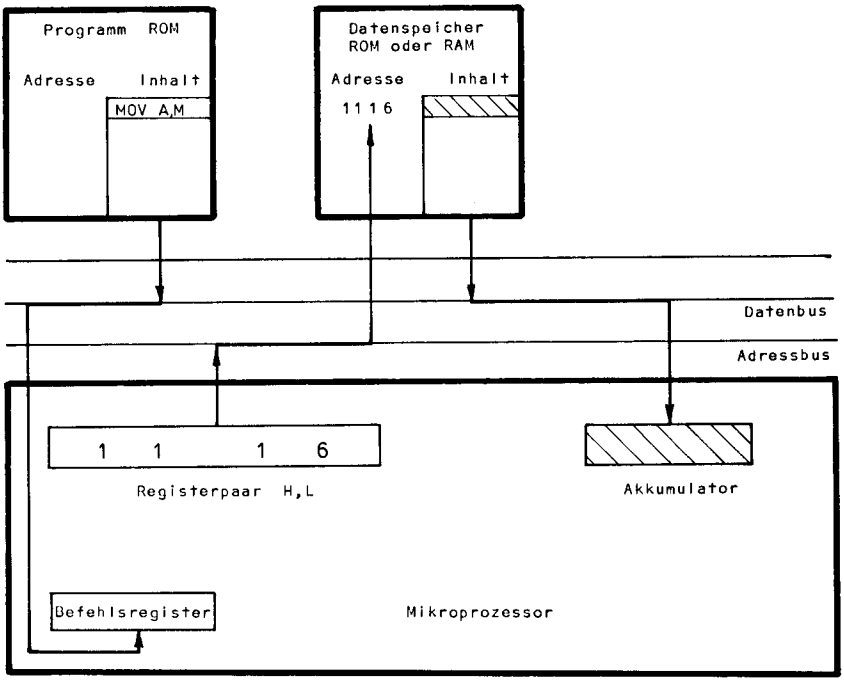


Bild 96: Registerpaar-Adressierung des 8080/8085

Beispiel: Laden des Registerpaares B,C - adressiert mit B - mit der Zahlenkonstanten $FFFF_{16}$

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	01		LXI	B,0FFFFH
	FF			
	FF			

Beispiel: Laden des Registerpaares B,C - adressiert mit B - mit der Adresse der Speicherstelle EVA.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000		EVA	DS	10
			ORG	2880H
2880	01		LXI	B,EVA
	00			
	20			

Der LXI-Befehl kann jedoch nicht den Inhalt einer Speicherstelle in ein Registerpaar laden. Dies ist nur für das Registerpaar H,L mit dem Befehl LHLD (Lade Registerpaar H,L Direct) möglich, der im 2. und 3. Byte eine Datenadresse enthält, deren Inhalt in das Registerpaar übertragen wird. Der MOV-Befehl verbindet die Register H und L mit den anderen Registerpaaren.

Beispiel: Laden des Registerpaares H,L mit der Konstanten $FFFF_{16}$, die in den Speicherstellen MAX und MAX+1 abgelegt wurde.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2000H
2000    FF      MAX    DW       0FFFFH
        FF

                       ORG      2880H
2880    2A             LHLD     MAX
        00
        20
```

Beispiel: Laden des Registerpaares H,L mit der Anfangsadresse des Bereiches OTTO, die in den Speicherstellen EMIL und EMIL+1 abgelegt wurde.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2000H
2000            OTTO   DS       10
200A    00      EMIL   DW       OTTO
        20

                       ORG      2880H
2880    2A             LHLD     EMIL
        0A
        20
```

Der Befehl SHLD (Speichere Registerpaar H,L Direct) speichert den Inhalt des Registerpaares H,L unter der angegebenen Datenadresse ab.

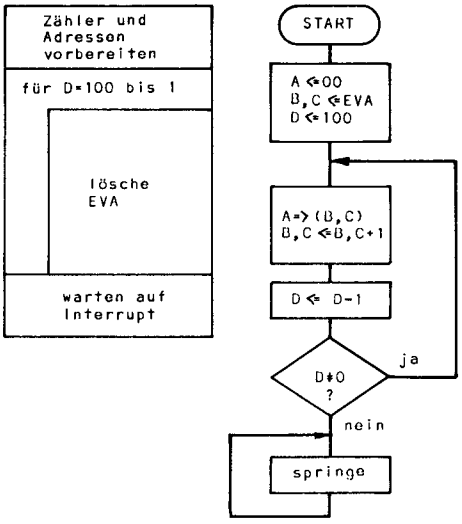
Die folgenden Befehle laden bzw. speichern den Akkumulator. Die Datenadresse steht in einem der beiden Registerpaare B,C bzw. D,E. Es sind 1-Byte-Befehle, der Name des Registerpaares ist im Funktionscode enthalten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H -P-C	
LDAX	B	Registerpaar	0A	1	7		Lade Akku mit Speicher, Adresse in B,C
LDAX	D	Registerpaar	1A	1	7		Lade Akku mit Speicher, Adresse in D,E
STAX	B	Registerpaar	02	1	7		Speichere Akku nach Speicher, Adr. B,C
STAX	D	Registerpaar	12	1	7		Speichere Akku nach Speicher, Adr. D,E

Das X im Befehl bedeutet, daß die Datenadresse in einem IndeX-register steht.

Lädt man ein Registerpaar mit der Anfangsadresse eines Bereiches und erhöht es anschließend in einer Schleife, so kann man alle Bytes des Bereiches nacheinander ansprechen. Da die Registerpaare nicht direkt mit einem Endwert verglichen werden können, ist es oft vorteilhafter, in einem anderen Register einen Zähler mitlaufen zu lassen.

Beispiel: Es ist ein Bereich von 100 Bytes von der Anfangsadresse EVA ab mit binären Nullen zu füllen. Das Registerpaar B,C wird mit der Anfangsadresse geladen und aufwärts gezählt; das Register D kontrolliert die Schleifendurchläufe.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000		EVA	DS	100
			ORG	2880H
2880	3E	START	MVI	A,00H
	00			
2882	01		LXI	B,EVA
	00			
	20			
2885	16		MVI	D,64H
	64			
2887	02	OTTO	STAX	B
2888	03		INX	B
2889	15		DCR	D
288A	C2		JNZ	OTTO
	87			
	28			
288D	C3	SUSI	JMP	SUSI
	8D			
	28			
			END	

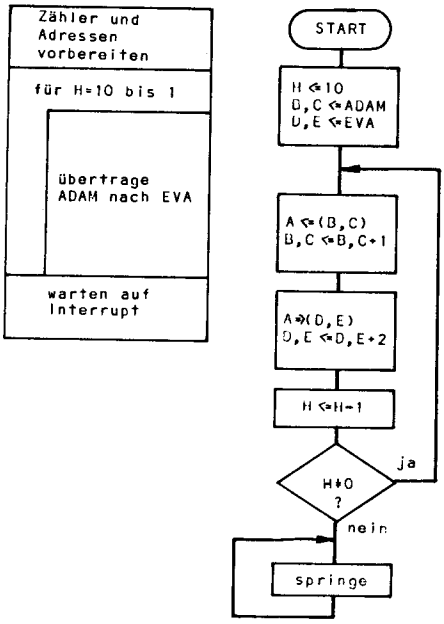
Die folgende Tabelle zeigt dezimal die Adreßrechnung.

D-Register Zähler	Datenadresse Registerpaar	Datenadresse symbolisch
100	2000	EVA+0
99	2001	EVA+1
98	2002	EVA+2
⋮	⋮	⋮
3	2097	EVA+97
2	2098	EVA+98
1	2099	EVA+99

Das Programm ohne Warteschleife besteht aus 13 Bytes, es wird in 30 + 100X17 = 1730 Zyklen ausgeführt. Ein entsprechendes Programm mit einem Lade- und 100 Speicherbefehlen besteht aus 2 + 100X3 = 302 Bytes und benötigt 10 + 100X13 = 1310 Zyklen. Die Indexschleife vermindert die Zahl der Befehle und verlängert die Laufzeit.

Sollen Daten von einem Speicherbereich in einen anderen übertragen werden, so sind zwei Registerpaare zur Indizierung zu verwenden.

Beispiel: Das folgende Programm überträgt 10 hintereinanderliegende Bytes aus einem Bereich ADAM (Anfangsadresse 2000) in jedes zweite Byte des Bereiches EVA (Anfangsadresse 200A). Das H-Register zählt die Schleifendurchläufe.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000		ADAM	DS	10
200A		EVA	DS	20
			ORG	2880H
2880	3E	START	MVI	H,10
	0A			
2882	01		LXI	B,ADAM
	00			
	20			
2885	11		LXI	D,EVA
	0A			
	20			
2888	0A	OTTO	LDAX	B
2889	03		INX	B
288A	12		STAX	D
288B	13		INX	D
288C	13		INX	D
288D	25		DCR	H
288E	C2		JNZ	OTTO
	88			
	28			
2891	C3	MAX	JMP	MAX
	91			
	28			
			END	

Die folgende Tabelle zeigt die Adreßrechnung

Zähler H-Register	Datenadresse B,C-Register	Datenadresse symbolisch	Datenadresse D,E-Register	Datenadresse symbolisch
10	2000	ADAM+0	200A	EVA+0
9	2001	ADAM+1	200C	EVA+2
8	2002	ADAM+2	200E	EVA+4
7	2003	ADAM+3	2010	EVA+6
6	2004	ADAM+4	2012	EVA+8
5	2005	ADAM+5	2014	EVA+10
4	2006	ADAM+6	2016	EVA+12
3	2007	ADAM+7	2018	EVA+14
2	2008	ADAM+8	201A	EVA+16
1	2009	ADAM+9	201C	EVA+18

Während die Registerpaare B,C und D,E nur für Befehle LDAX (Lade) und STAX (Speichere) verwendet werden können, kann das Registerpaar H,L für eine Reihe weiterer Befehle die Datenadresse aufnehmen. Diese Befehle enthalten im Operandenteil den Kennbuchstaben M für Memory = Speicher und sprechen die Datenspeicherstelle an, deren Adresse im Registerpaar H,L steht.

Laden eines Registers

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H -P-C	
MOV	A,M	Registerpaar	7E	1	7		Lade Akku mit Speicher, Adresse in H,L
MOV	B,M	Registerpaar	46	1	7		Lade B-Register mit Speicher, Adr. H,L
MOV	C,M	Registerpaar	4E	1	7		Lade C-Register mit Speicher, Adr. H,L
MOV	D,M	Registerpaar	56	1	7		Lade D-Register mit Speicher, Adr. H,L
MOV	E,M	Registerpaar	5E	1	7		Lade E-Register mit Speicher, Adr. H,L
MOV	H,M	Registerpaar	66	1	7		Lade H-Register mit Speicher, Adr. H,L
MOV	L,M	Registerpaar	6E	1	7		Lade L-Register mit Speicher, Adr. H,L

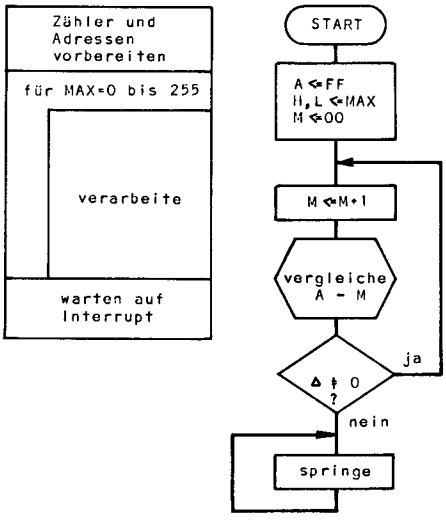
Speichern eines Registers = Laden einer Speicherstelle

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
MOV	M,A	Registerpaar	77	1	7		Lade Speicher mit Akkumulator, Adr. in H,L
MOV	M,B	Registerpaar	70	1	7		Lade Speicher mit B-Register, Adresse H,L
MOV	M,C	Registerpaar	71	1	7		Lade Speicher mit C-Register, Adresse H,L
MOV	M,D	Registerpaar	72	1	7		Lade Speicher mit D-Register, Adresse H,L
MOV	M,E	Registerpaar	73	1	7		Lade Speicher mit E-Register, Adresse H,L
MOV	M,H	Registerpaar	74	1	7		Lade Speicher mit H-Register, Adresse H,L
MOV	M,L	Registerpaar	75	1	7		Lade Speicher mit L-Register, Adresse H,L

Laden, Zählen und Vergleichen einer Speicherstelle

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
MVI	M,konst	Registerpaar	36	2	7		Lade Speicher mit 1-Byte-Konst, Adresse H,L
INR	M	Registerpaar	34	1	10	XX X X	Erhöhe Speicher um 1, Adresse in H,L
DCR	M	Registerpaar	35	1	10	XX X X	Vermindere Speicher um 1, Adresse H,L
CMP	M	Registerpaar	BE	1	7	XX X X X	Vergleiche Akku - Speicher, Adresse H,L

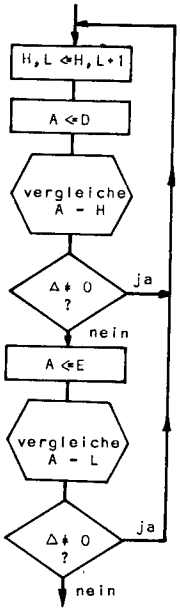
Beispiel: In der Speicherstelle MAX soll unter Verwendung von Befehlen mit Registerpaar-Adressierung ein Zähler von Null bis zum Endwert FF laufen.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2000H
2000		MAX	DS	1
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	21		LXI	H,MAX
	00			
	20			
2885	36		MVI	M,00
	00			
2887	34	OTTO	INR	M
2888	BE		CMP	M
2889	C2		JNZ	OTTO
	87			
	28			
288C	C3	MAUS	JMP	MAUS
	8C			
	28			
			END	

Das Registerpaar H,L, das bei der Registerpaar-Adressierung die laufende Datenadresse enthält, läßt sich nicht durch einen einzigen Vergleichsbefehl auf einen Endwert abfragen. Dies kann nur mit Hilfe des Akkumulators geschehen.

Beispiel: Das Registerpaar H,L ist auf einen Endwert abzufragen, der im Registerpaar D,E steht.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2890H
2890	23	MAX1	INX	H
2891	7A		MOV	A,D
2892	BC		CMP	H
2893	C2		JNZ	MAX1
	90			
	28			
2896	7B		MOV	A,E
2897	BD		CMP	L
2898	C2		JNZ	MAX1
	90			
	28			

4.6 Die indirekte Adressierung

A

In den bisher behandelten Aufgaben war die Datenadresse bei der Aufstellung des Programms bekannt. Entweder stand sie im Adreßteil des Befehls oder sie wurde über ein Indexregister verändert. Es gibt jedoch Anwendungen, bei denen die Datenadresse entweder von außen eingegeben oder im Programm berechnet wird oder bei denen die Datenadresse je nach Hauptprogramm wechselt, wie z.B. bei Unterprogrammen. In diesen Fällen enthält der Befehl die Adresse einer Speicherstelle, in der die richtige Datenadresse zu finden ist. Bei dieser indirekten Adressierung wird also der Befehl nicht mit der im Befehl angegebenen Adresse, sondern mit der Adresse ausgeführt, die dort abgespeichert ist. Dieser Vorgang heißt auch Adreß-Substitution, weil die im Befehl stehende Adresse durch den Inhalt der angesprochenen Speicherstelle ersetzt wird. Bild 97 zeigt einen Befehl mit indirekter Adressierung. Die in einem Ladebefehl enthaltene indirekte Adresse 1111 weist auf eine Speicherstelle 2222 hin, deren Inhalt in den Akkumulator zu bringen ist.

Der in Bild 97 dargestellte Ladebefehl wird in folgenden Schritten ausgeführt:

1. Funktionscode holen
2. indirekte Adresse holen
3. indirekte Adresse aussenden
4. richtige Datenadresse holen
5. richtige Datenadresse aussenden
6. Befehl mit richtiger Datenadresse ausführen

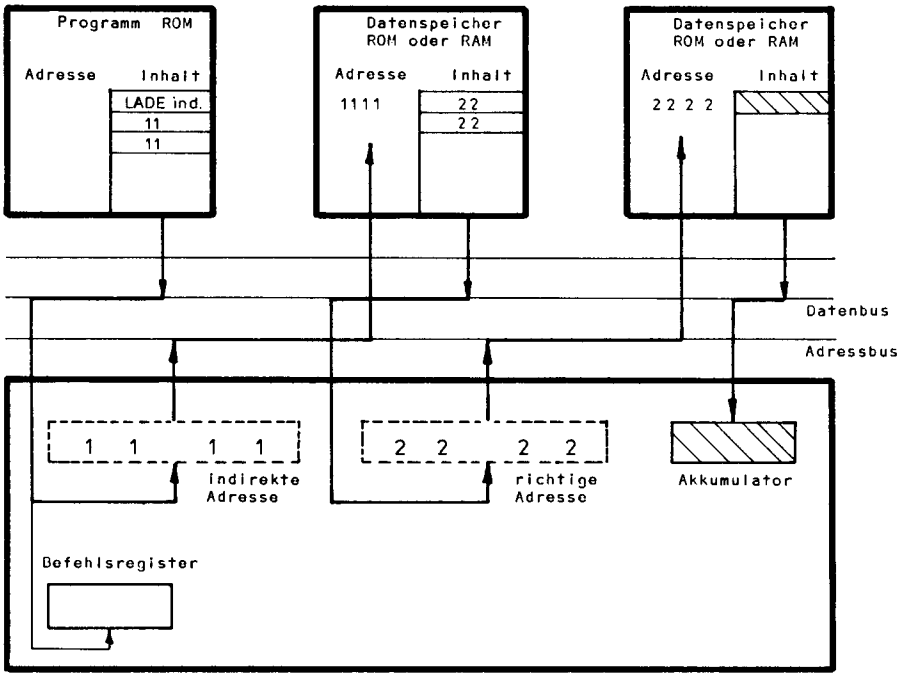


Bild 97: Schema der indirekten Adressierung

4.6.1 Die indirekte Adressierung des KIM-1 KIM

Der KIM-1 verwendet die indirekte Adressierung nur zusammen mit den beiden Indexregistern.

a. Die indiziert-indirekte Adressierung (Bild 98)

Die im Befehl enthaltene indirekte Adresse ist ein byte lang und muß daher in Seite Null (zero-page) liegen. Sie wird aus dem Programmspeicher geholt, zum Inhalt des X-Registers addiert und ergibt eine neue zero-page Adresse von acht bit Länge. Die hiermit adressierte Speicherstelle enthält das niederwertige Byte, die folgende Speicherstelle enthält das höherwertige Byte der richtigen Datenadresse.

Das in Bild 98 dargestellte Beispiel wird in folgenden Schritten ausgeführt:

1. Funktionscode in das Befehlsregister bringen
2. Indirekte Adresse 11 holen
3. Indirekte Adresse 11 zum Inhalt des X-Registers 05 addieren
4. Neue indirekte Adresse 0016 aussenden
5. Niederwertiges Byte der richtigen Adresse 15 holen
6. Um 1 erhöhte indirekte Adresse 0017 aussenden
7. Höherwertiges Byte der richtigen Adresse 22 holen
8. Richtige Datenadresse 2215 aussenden
9. Inhalt der Speicherstelle 2215 in den Akkumulator bringen

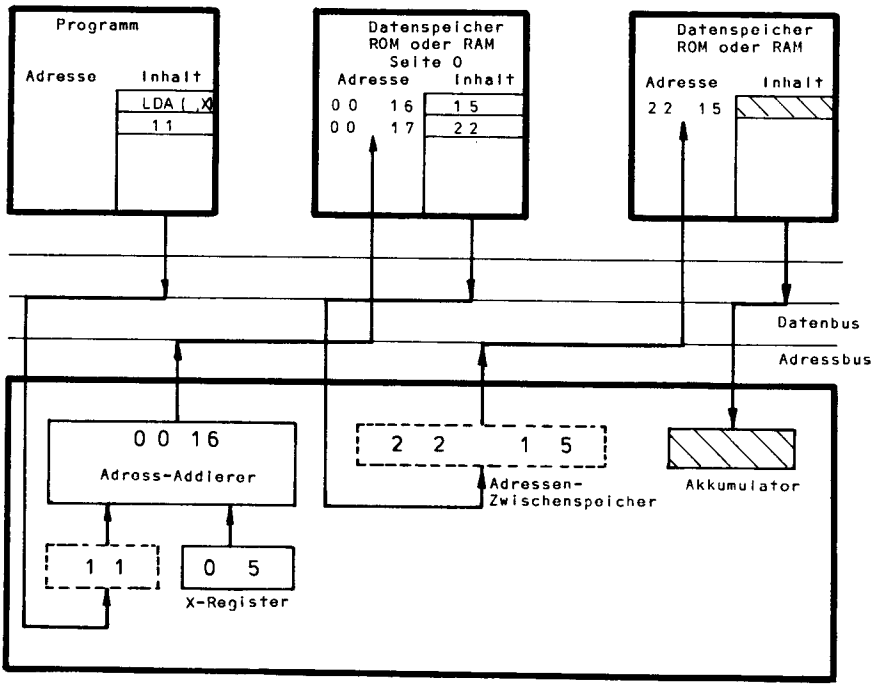


Bild 98: Indiziert-indirekte Adressierung des KIM-1

Ist die Summe aus indirekter Adresse plus Inhalt des X-Registers größer als 255, so wird der Übertrag nicht berücksichtigt, das höherwertige Byte der Adresse wird immer 00 gesetzt. Dies kann unter Umständen zu Fehlern führen.

Die indiziert-indirekte Adresse (erst indiziert, dann indirekt) kann auf die folgenden bisher behandelten Befehle angewendet werden:

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DI ZC	
LDA	(Adr,X)	indiz-indir.	A1	2	6	X X	Lade Akkumulator indiziert-indirekt
STA	(Adr,X)	indiz-indir.	81	2	6		Speichere Akkumulator indiziert-indirekt
CMP	(Adr,X)	indiz-indir	C1	2	6	X XX	Vergleiche Akkumulator indiziert-indirekt

Der in Klammern eingeschlossene Adreßteil bedeutet indirekte Adressierung, das X in der Klammer weist auf die vorhergehende Indizierung hin.

Beispiel: Der Akkumulator soll mit dem Inhalt einer Speicherstelle MAUS geladen werden, deren Adresse in der Speicherstelle KATZE steht. Die Adresse von KATZE wird nicht verändert, das X-Register sei Null.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name    Befehl    Operand

                        *=$0000
0000      FF    MAUS    .BYTE     $FF
0001      00    KATZE   .WORD     MAUS
          00

                        *=$0300
0300      A2            LDX       #$00
          00
0302      A1            LDA       (KATZE,X)
          01
0304
```

Mit Hilfe des X-Registers ist es möglich, unter verschiedenen indirekten Adressen eine bestimmte auszuwählen.

b. Die indirekt-indizierte Adressierung (Bild 99)

Die im Befehl enthaltene indirekte Adresse ist ein byte lang und adressiert eine in Seite Null (zero-page) liegende Speicherstelle, die das niederwertige Byte einer Datenadresse enthält. Die folgende Speicherstelle enthält das höherwertige Byte. Zu dieser Adresse wird noch der Inhalt des Y-Registers addiert. Die Summe ergibt die richtige Datenadresse, mit der der Befehl ausgeführt wird.

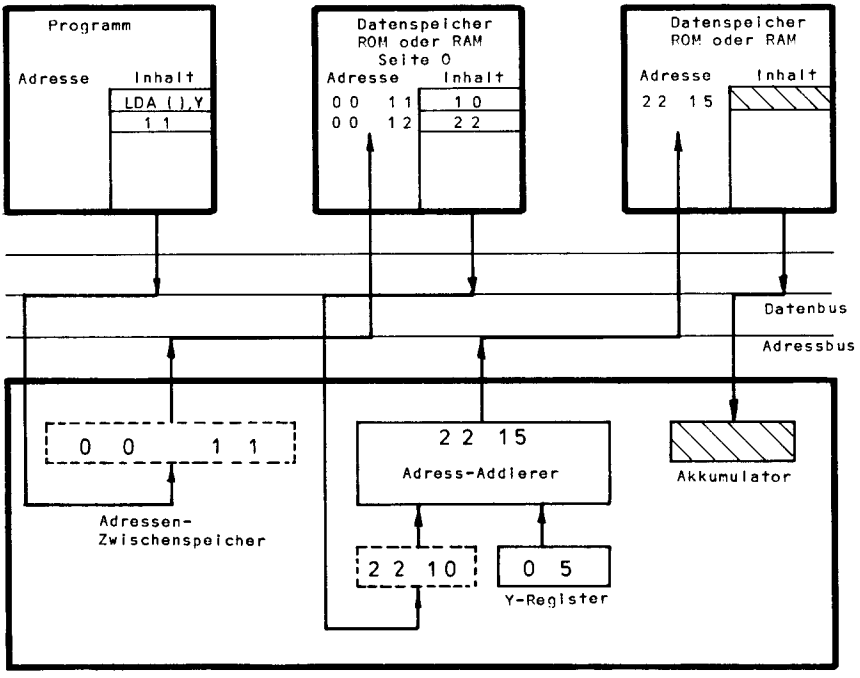


Bild 99: Indirekt-indizierte Adressierung des KIM-1

Das in Bild 99 dargestellte Beispiel wird in folgenden Schritten ausgeführt:

1. Funktionscode in das Befehlsregister bringen
2. Indirekte Adresse 11 holen

3. Indirekte Adresse 0011 aussenden
4. Niederwertiges Byte der richtigen Adresse 10 holen
5. Inhalt des Y-Registers 05 dazuaddiert gibt 15
6. Um 1 erhöhte indirekte Adresse 0012 aussenden
7. Höherwertiges Byte der richtigen Adresse 22 holen und eventuell Übertrag addieren
8. Richtige Datenadresse 2215 aussenden
9. Inhalt der Speicherstelle 2215 in den Akkumulator bringen

Die indirekt-indizierte Adressierung (erst indirekt, dann indiziert) kann auf die folgenden bisher behandelten Befehle angewendet werden:

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	V	B	D	I	Z	C	
LDA	(Adr),Y	indir-indiz.	B1	2	5/6	X					X		Lade Akkumulator indirekt-indiziert
STA	(Adr),Y	indir-indiz.	91	2	5/6								Speichere Akkumulator indirekt-indiziert
CMP	(Adr),Y	indir-indiz.	D1	2	5/6	X					X	X	Vergleiche Akkumulator indirekt-indiziert

Die in Klammern gesetzte Adresse bedeutet indirekte Adressierung, das Y dahinter weist auf die folgende Indizierung hin.

Die Befehle dauern fünf Zyklen ohne Seitenüberschreitung bzw. sechs Zyklen mit Seitenüberschreitung bei der Adreßrechnung. Mit Hilfe des Y-Registers ist es möglich, die richtige Datenadresse in einer Schleife zu verändern.

Beispiel: Der Bereich MAUS besteht aus 10 Bytes, die auszugeben sind. Die Speicherstelle KATZE enthält die Adresse von MAUS.

```
  Hexadezimal          Symbolisch
Adresse Inhalt   Name    Befehl    Operand

                 AUS     =$1700

                         *=$0000
0000             MAUS    *=*+10
000A     00      KATZE   .WORD     MAUS
         00

                         *=$0300
0300     A0              LDY       #$09
         09
0302     B1      MAX     LDA       (KATZE),Y
         0A
0304     8D              STA       AUS
         00
         17
0307     88              DEY
0308     30              BMI       MAX
         F8
030A
```

Alle im Befehl stehenden indirekten Adressen müssen in Seite Null liegen! Mit Hilfe des Y-Registers ist es möglich, die endgültige Datenadresse zu verändern.

4.6.2 Indirekte Adressierung mit dem 6800 **6800**

Da der Mikroprozessor 6800 keine eingebaute indirekte Adressierung hat, muß vorher das Indexregister mit der Adresse der Speicherstelle geladen werden, mit der der Befehl auszuführen ist. Das folgende Beispiel Bild 100 zeigt, wie zunächst das Indexre-

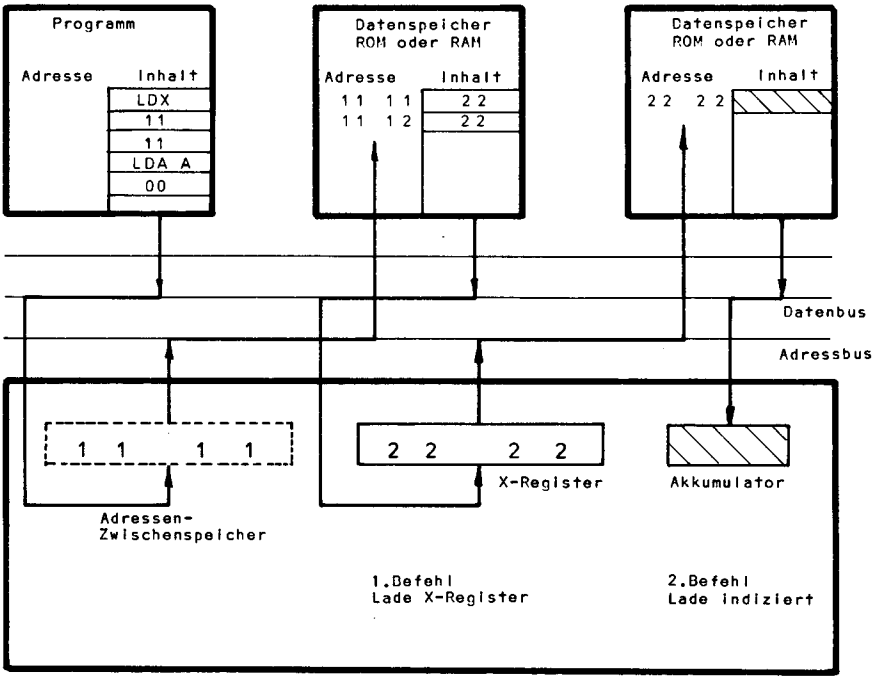


Bild 100: Indirekte Adressierung mit dem 6800

gister mit dem Inhalt der Speicherstelle 1111 geladen wird. Dies ist die neue Adresse 2222. Anschließend bringt ein Ladebefehl mit indizierter Adressierung den Inhalt der Speicherstelle 2222 in den Akkumulator.

Für die indirekte Adressierung beim 6800 sind zwei Befehle erforderlich:
1. Laden des Indexregisters mit einer Adresse
2. Speicherzugriff mit indizierter Adressierung

Beispiel: Der Akkumulator soll mit dem Inhalt der Speicherstelle MAUS geladen werden. Die Adresse von MAUS steht in KATZE.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000      FF    MAUS   FCB      $FF
0001      00    KATZE  FDB      MAUS
          00

                       ORG      $0180
0180      FE             LDX      KATZE
          00
          01
0183      A6             LDA A    00,X
          00
0185
```

4.6.3 Indirekte Adressierung mit dem 8080/8085

Die Mikroprozessoren 8080 und 8085 haben keine eingebaute indirekte Adressierung. Daher muß zunächst ein Registerpaar mit der Adresse der Speicherstelle geladen werden.
Das folgende Beispiel Bild 101 zeigt, wie zunächst das Registerpaar H,L mit dem Inhalt der Speicherstelle 1111, nämlich mit der richtigen Datenadresse geladen wird. Anschließend bringt ein Ladebefehl mit Registerpaar-Adressierung den Inhalt der Speicherstelle 2222 in den Akkumulator.

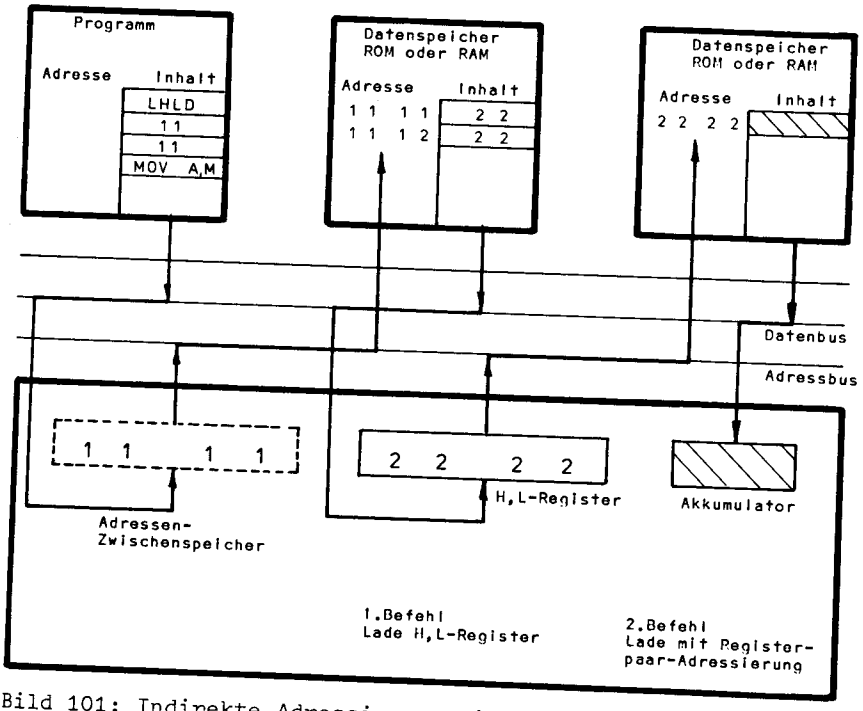


Bild 101: Indirekte Adressierung mit dem 8080/8085

Für die indirekte Adressierung beim 8080/8085 sind zwei Befehle erforderlich:
1. Laden eines Registerpaares mit einer Adresse
2. Speicherzugriff mit Registerpaar-Adressierung

Beispiel: Der Akkumulator ist mit dem Inhalt der Speicherstelle MAUS zu laden. Die Adresse von MAUS ist in der Speicherstelle KATZE abgelegt.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
2000	FF	MAUS	ORG	2000H
2001	00	KATZE	DB	OFFH
	20		DW	MAUS
2880	2A		ORG	2880H
	01		LHLD	KATZE
	20			
2883	7E		MOV	A,M
2884				

Übungen zum Abschnitt 4.5. und 4.6.

1.Aufgabe

Bei der Entwicklung von Mikrocomputern und Zusatzspeichern müssen Schreib/Lese-Speicher (RAMs) geprüft werden. Es ist ein Programm zu entwerfen, das einen RAM-Bereich von 256 Bytes (eine Seite) mit den Dualzahlen von 0 bis 255 beschreibt, anschließend wieder liest und mit einem Zähler vergleicht. Tritt keine Abweichung auf, so bleibt das erste Byte des Bereiches Null, sonst enthält es die Zahl der Fehler.

2.Aufgabe

An einer Parallelschnittstelle liegt eine Ein/Ausgabe-Schaltung nach Bild 48. Über die acht Kippschalter wird ein Wert von 00 bis FF eingegeben. Dies ist das niederwertige Byte einer Datenadresse, deren höherwertiges Byte im Programm festgelegt wurde. Der Inhalt der damit adressierten Speicherstelle ist anzuzeigen.

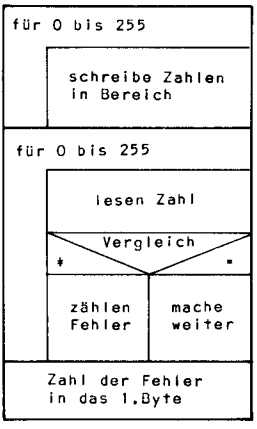
Beispiel:	Basisadresse im Programm	0100
	eingegebene Adresse	FA
	auszugebende Speicherstelle	01FA

3.Aufgabe

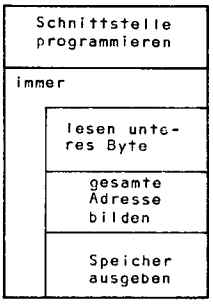
Es ist ein beliebiger Speicherbereich zu löschen, Anfangs- und Endadresse sind in zwei Speicherstellen vor dem Programm abzulegen. Anschließend soll das Programm in einer unendlichen Schleife auf einen Abbruch warten.

Programmblockpläne

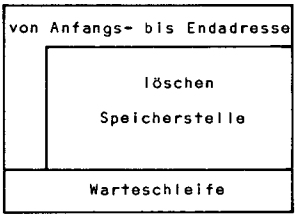
1.Aufgabe



2.Aufgabe



3.Aufgabe



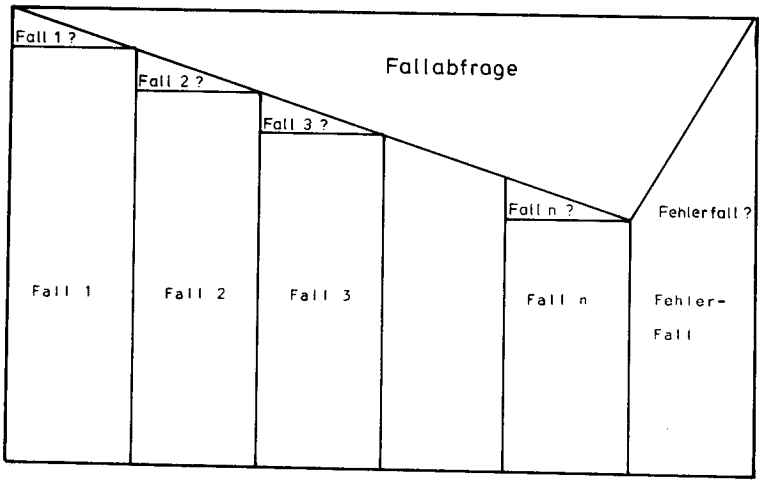
5. Programmverzweigungen

A

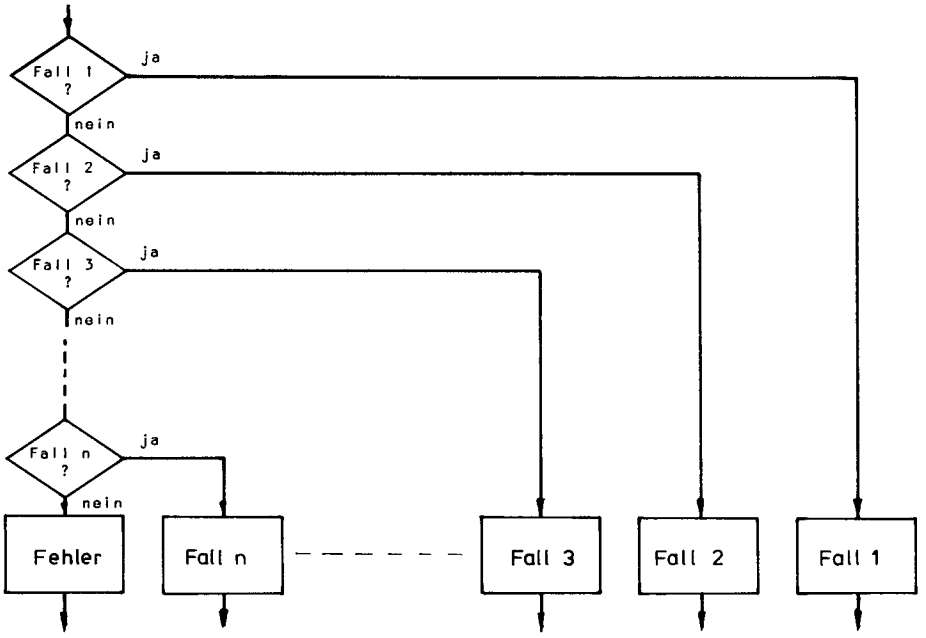
Dieser Abschnitt behandelt Programmverzweigungen als Anwendung bedingter Sprünge.

Beispiel: Mikrocomputer-Programme haben häufig die Aufgabe, die Stellung von Eingabe-Schaltern abzufragen. So liege an einer Parallelschnittstelle eine Eingabeschaltung mit acht Kippschaltern nach Bild 48. Jeder Schalter steuere ein anderes Ausgabeprogramm. Schalter 7 = 1 erzeuge eine Rechteckspannung von 1 kHz, Schalter 6 = 1 lasse einen Sekundenzähler laufen, Schalter 5 = 1 erzeuge eine Blinkfrequenz von 1 Hz usw. Für den Fall, daß zwei Schalter gleichzeitig 1 sind, kann entweder eine Fehleranzeige ausgegeben werden oder es kann vereinbart werden, daß die höherwertigen Schalter zu bevorzugen sind.

Der Programmblockplan stellt diese Fallunterscheidung allgemein durch folgendes Symbol dar:

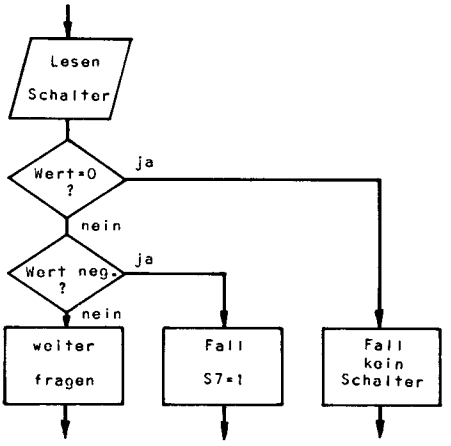


Der Programmablaufplan und damit das Programm bestehen aus einer Folge von einzelnen Verzweigungen, die recht unübersichtlich werden können. Der Programmblockplan zeigt nur die Struktur des Programms und berücksichtigt nicht, wie die Verzweigungen im Einzelnen ausgeführt werden.

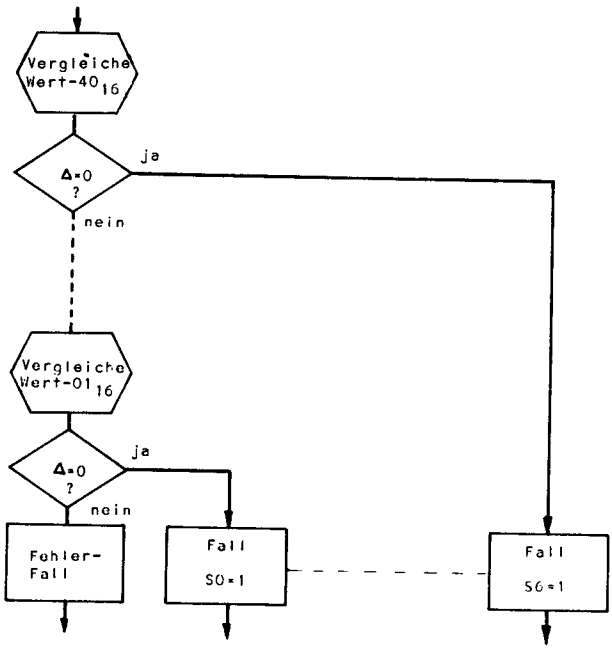


5.1 Verzweigungen mit Sprung- und Vergleichsbefehlen

Die bedingten Sprünge bzw. Verzweigungsbefehle lassen nur eine Abfrage auf Null oder auf Vorzeichen (werthöchstes Bit Pos.7) zu. In dem eingangs erwähnten Beispiel können also nur die Fälle "kein Schalter" und "Schalter 7" sofort entschieden werden.



Zur Abfrage der restlichen Schalterstellungen sind Vergleichsbefehle erforderlich, die vom Inhalt eines Registers einen Vergleichswert subtrahieren und mit dem Ergebnis die Bedingungen des Bedingungsregisters beeinflussen, ohne den Inhalt des Registers mit der Differenz zu zerstören.



Die Vergleichsbefehle fragen ein ganzes 8-Bit-Wort (Byte) ab. Nur über umständliche Bereichsabfragen ist es möglich, den Wert eines einzelnen Bits unabhängig von den anderen zu bestimmen.

5.1.1 Verzweigungen mit Vergleichsbefehlen des KIM-1 **KIM**

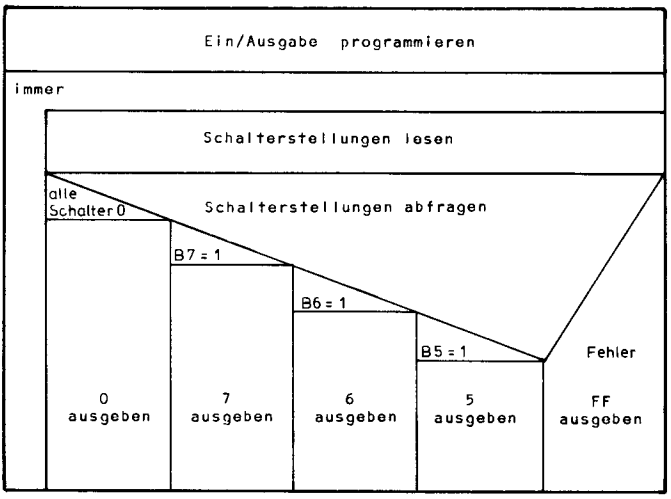
Die Verzweigungs- und Vergleichsbefehle wurden bereits im Abschnitt 4 behandelt. Der BIT-Befehl bringt Bit 7 einer Speicherstelle in das N-Bit (Negativ) und Bit 6 in das V-Bit (Overflow) des Bedingungsregisters. Damit ist Bit 6 direkt abfragbar. Zusätzlich wird das logische UND des Akkumulators und des Speicherinhaltes gebildet und mit dem Ergebnis das Z-Bit (Zero) verändert. Ist das logische Produkt aller acht Bitpositionen Null, so wird das Z-Bit 1, sonst 0. Lädt man vor dem BIT-Befehl den Akkumulator mit der Konstanten FF, so gibt das Z-Bit an, ob die untersuchte Speicherstelle Null ist (Z=1) oder ob sie nicht Null ist (Z=0).

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N V B D I Z C	
BIT	Adresse	zero-page	24	2	3	XX X	N=Bit7 , V=Bit6 , Z=1 für (Akku AND Sp.)=0
BIT	Adresse	absolut	2C	3	4	XX X	N=Bit7 , V=Bit6 , Z=1 für (Akku AND Sp.)=0

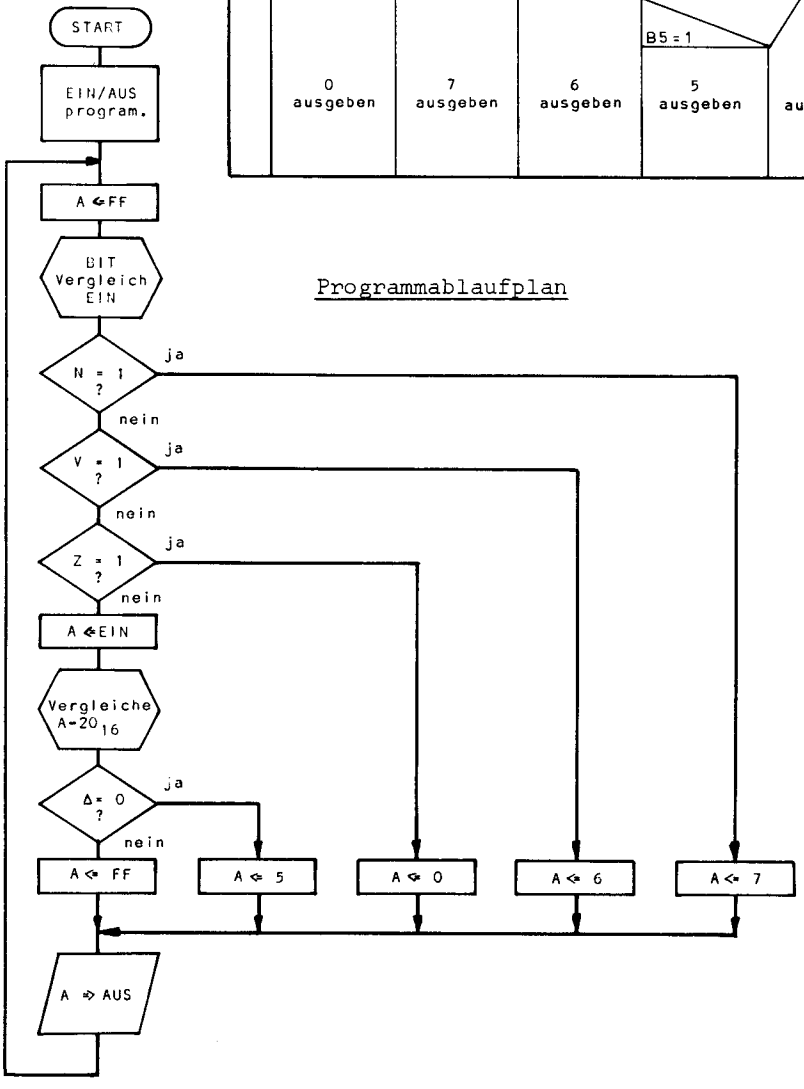
Beispiel:

An einer Parallelschnittstelle liege eine Eingabeschaltung nach Bild 48 mit acht Kippschaltern, von denen nur die drei werthöchsten verwendet werden, die restlichen fünf seien ohne Bedeutung.

Programmblockplan



Programmablaufplan



Programm

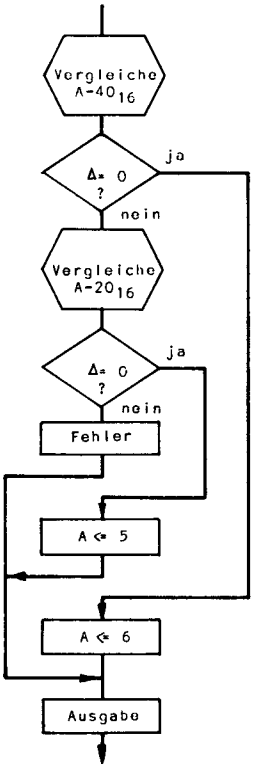
Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	=$1700	
		RAU	=$1701	
		EIN	=$1702	
		REI	=$1703	
			*=$0300	
0300	A9	START	LDA	#$00
	00			
0302	8D		STA	REI
	03			
	17			
0305	A9		LDA	#$FF
	FF			
0307	8D		STA	RAU
	01			
	17			
030A	A9	ANF	LDA	#$FF
	FF			
030C	2C		BIT	EIN
	02			
	17			
030F	30		BMI	M7
	1F			
0311	70		BVS	M6
	18			
0313	F0		BEQ	M0
	0C			
0315	AD		LDA	EIN
	02			
	17			
0318	C9		CMP	#$20
	20			
031A	F0		BEQ	M5
	0A			
031C	A9		LDA	#$FF
	FF			
031E	4C		JMP	ANZ
	32			
	03			
0321	A9	M0	LDA	#$00
	00			
0323	4C		JMP	ANZ
	32			
	03			
0326	A9	M5	LDA	#$05
	05			
0328	4C		JMP	ANZ
	32			
	03			
032B	A9	M6	LDA	#$06
	06			
032D	4C		JMP	ANZ
	32			
	03			
0330	A9	M7	LDA	#$07
	07			
0332	8D	ANZ	STA	AUS
	00			
	17			
0335	4C		JMP	ANF
	0A			
	03			
			.END	

Für Schalter B7 = 1 werde die Zahl 7 dual angezeigt,
für Schalter B6 = 1 werde die Zahl 6 dual angezeigt und
für Schalter B5 = 1 werde die Zahl 5 dual angezeigt.
Sind alle Schalterstellungen Null, so sei die Ausgabe gelöscht.
Ist mehr als ein Schalter 1, so sollen als Fehlermeldung alle Anzeigen leuchten.

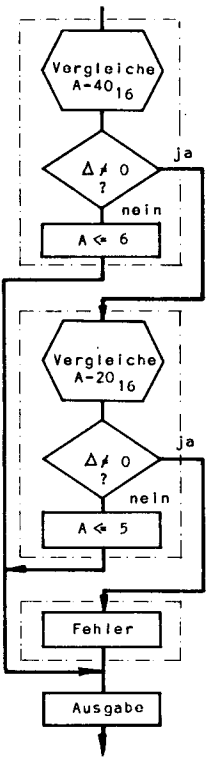
Die Aufgabe wird hier mit Programmverzweigungen gelöst, obwohl es auch andere Möglichkeiten gibt, wie z.B. das Suchen der Schalterstellungen in einer Tabelle. Man beachte, daß auf der B-Seite der Parallelschnittstelle die Leitung B6 nicht angeschlossen ist. Im Programm wird dies jedoch nicht berücksichtigt.

Bei dem Beispiel liegen Abfrage und Behandlung eines Falls weit voneinander entfernt. Das Programm wird wesentlich übersichtlicher, wenn man den Fall sofort nach seiner Abfrage behandelt und bei "Bedingung nicht erfüllt" zu weiteren Abfragen springt. Das folgende Beispiel vergleicht beide Verfahren. Wie vorher sind die Fälle Akku = $40 und Akku = $20 zu untersuchen.

Verzweige bei
"Bedingung erfüllt"



Verzweige bei
"Bedingung nicht erfüllt"



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		AUS	=$1700	
			*=$0310	
0310	C9		CMP	#$40
	40			
0312	F0		BEQ	M6
	0E			
0314	C9		CMP	#$20
	20			
0316	F0		BEQ	M5
	05			
0318	A9		LDA	#$FF
	FF			
031A	4C		JMP	ANZ
	24			
	03			
031D	A9	M5	LDA	#$05
	05			
031F	4C		JMP	ANZ
	24			
	03			
0322	A9	M6	LDA	#$06
	06			
0324	8D	ANZ	STA	AUS
	00			
	17			

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		AUS	=$1700	
			*=$0310	
0310	C9		CMP	#$40
	40			
0312	D0		BNE	W5
	05			
0314	A9		LDA	#$06
	06			
0316	4C		JMP	ANZ
	24			
	03			
0319	C9	W5	CMP	#$20
	20			
031B	D0		BNE	FEHL
	05			
031D	A9		LDA	#$05
	05			
031F	4C		JMP	ANZ
	24			
	03			
0322	A9	FEHL	LDA	#$FF
	FF			
0324	8D	ANZ	STA	AUS
	00			
	17			

5.1.2 Verzweigungen mit Vergleichsbefehlen des 6800 **6800**

Die im Abschnitt 4.3.2. behandelten Verzweigungsbefehle (bedingte Sprünge) werten sowohl einzelne Bits des Bedingungsregisters (z.B. das Z-Bit) als auch ihre logischen Verknüpfungen für Programmverzweigungen aus. Am häufigsten werden die Abfragen für Negativ (N-Bit) und für Zero = Null (Z-Bit) benutzt.

Zur Abfrage auf einen beliebigen Wert in einem Register dienen die im Abschnitt 4.4.2. behandelten Vergleichsbefehle, die vom Inhalt eines Registers einen Vergleichswert subtrahieren und mit dem Ergebnis das Bedingungsregister beeinflussen, ohne jedoch den Inhalt des Registers zu verändern.

Ähnlich arbeiten die Testbefehle, die vom Inhalt eines Registers oder einer Speicherstelle den Wert 00 subtrahieren - also den Inhalt nicht verändern - aber dabei das Bedingungsregister beeinflussen. Damit ist es möglich, ein Eingaberegister auf Null bzw. Vorzeichen zu testen, ohne den Inhalt erst in ein Register des Mikroprozessors bringen zu müssen.

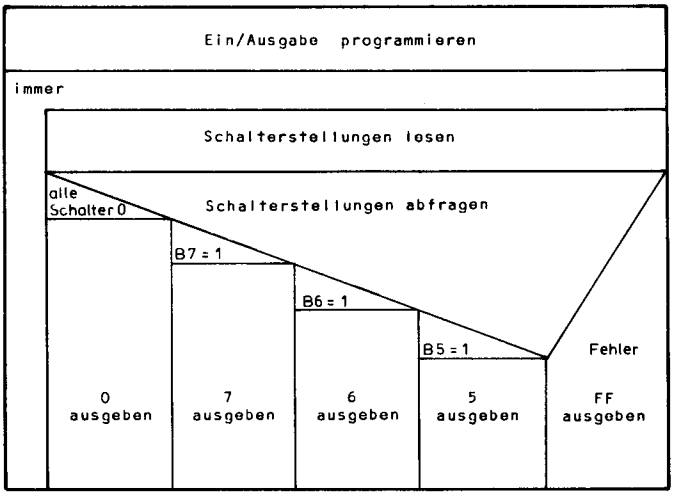
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen HI NZVC	
TST A		code-eigen	4D	1	2	XX00	Teste Akku A - 00
TST B		code-eigen	5D	1	2	XX00	Teste Akku B - 00
TST	Adresse	absolut	7D	3	6	XX00	Teste Speicher - 00 (extended adressiert)
TST	konst,X	indiziert	6D	2	7	XX00	Teste Speicher - 00 (indexed adressiert)

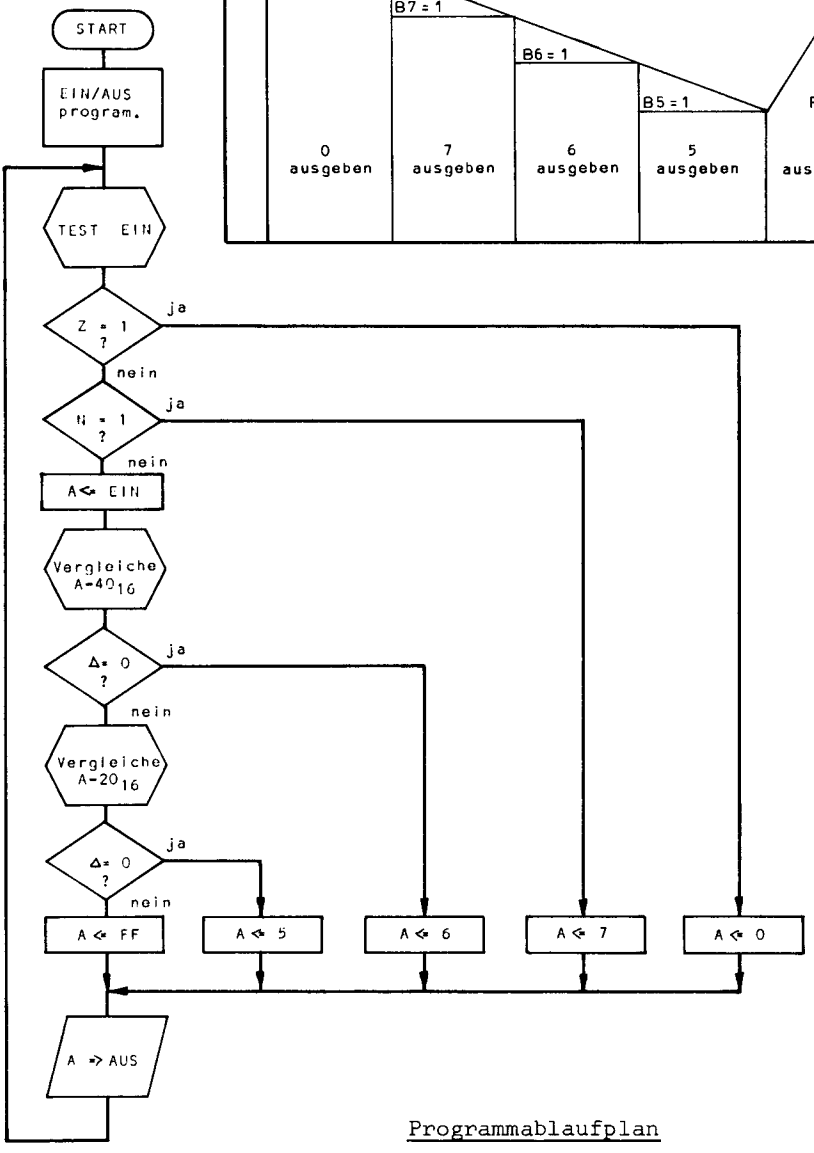
TST bedeutet TeSTe den Inhalt einer Speicherstelle oder eines Registers.

Beispiel:

An einer Parallelschnittstelle liege eine Eingabeschaltung nach Bild 48 mit acht Kippschaltern, von denen nur die drei werthöchsten verwendet werden, die restlichen fünf seien ohne Bedeutung.

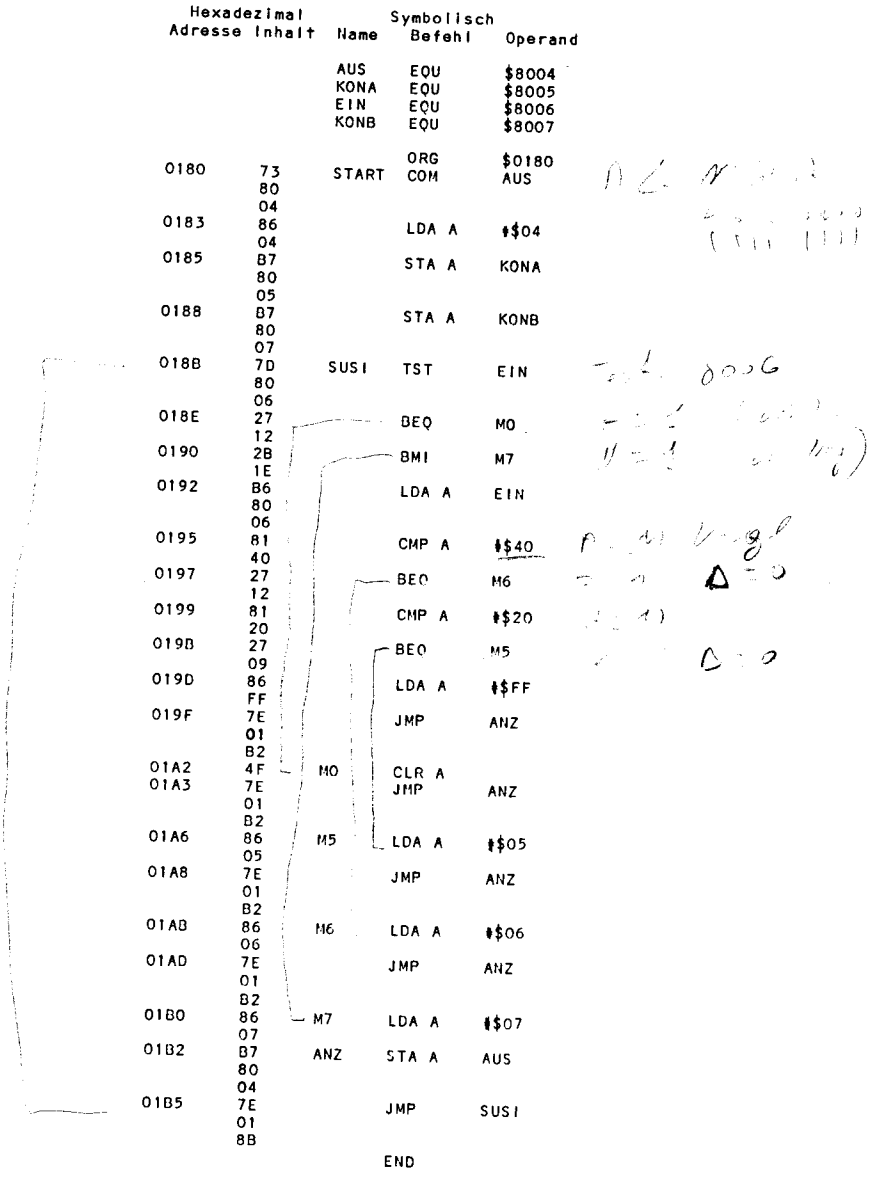
Programmblockplan





Programmablaufplan

Programm



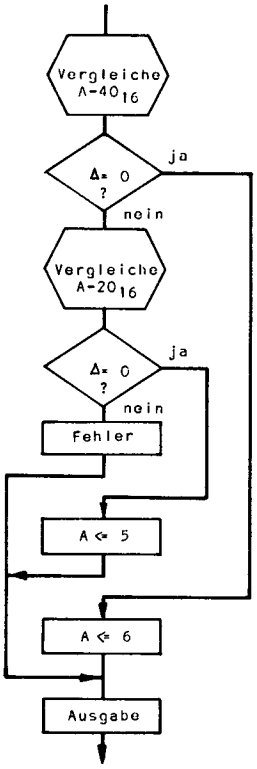
```
Hexadezimal                Symbolisch
Adresse Inhalt   Name      Befehl    Operand

                 AUS       EQU       $8004
                 KONA      EQU       $8005
                 EIN       EQU       $8006
                 KONB      EQU       $8007

                           ORG       $0180
0180    73       START     COM       AUS
        80
        04
0183    86                 LDA A     #$04
        04
0185    B7                 STA A     KONA
        80
        05
0188    B7                 STA A     KONB
        80
        07
018B    7D       SUSI      TST       EIN
        80
        06
018E    27                 BEQ       MO
        12
0190    2B                 BMI       M7
        1E
0192    B6                 LDA A     EIN
        80
        06
0195    81                 CMP A     #$40
        40
0197    27                 BEQ       M6
        12
0199    81                 CMP A     #$20
        20
019B    27                 BEQ       M5
        09
019D    86                 LDA A     #$FF
        FF
019F    7E                 JMP       ANZ
        01
        B2
01A2    4F       MO        CLR A
01A3    7E                 JMP       ANZ
        01
        B2
01A6    86       M5        LDA A     #$05
        05
01A8    7E                 JMP       ANZ
        01
        B2
01AB    86       M6        LDA A     #$06
        06
01AD    7E                 JMP       ANZ
        01
        B2
01B0    86       M7        LDA A     #$07
        07
01B2    B7       ANZ       STA A     AUS
        80
        04
01B5    7E                 JMP       SUSI
        01
        8B
                           END
```

Für Schalter B7 = 1 werde die Zahl 7 dual angezeigt,
für Schalter B6 = 1 werde die Zahl 6 dual angezeigt und
für Schalter B5 = 1 werde die Zahl 5 dual angezeigt.
Sind alle Schalterstellungen Null, so sei die Ausgabe gelöscht.
Ist mehr als ein Schalter 1, so sollen als Fehlermeldung alle Anzeigen leuchten.

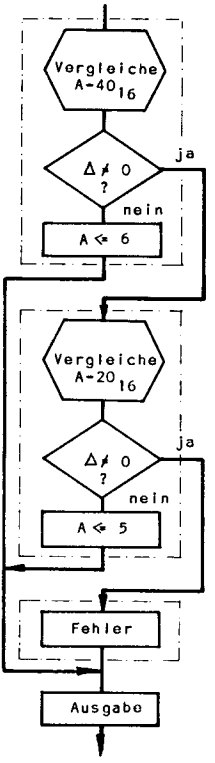
Die Aufgabe wird hier mit Programmverzweigungen gelöst, obwohl es auch andere Möglichkeiten gibt, wie z.B. das Suchen der Schalterstellungen in einer Tabelle.

Bei dem Beispiel liegen Abfrage und Behandlung eines Falls weit voneinander entfernt. Das Programm wird wesentlich übersichtlicher, wenn man den Fall sofort nach seiner Abfrage behandelt und bei "Bedingung nicht erfüllt" zu weiteren Abfragen springt. Das folgende Beispiel vergleicht beide Verfahren. Wie vorher sind die Fälle Akku = $40 und Akku = $20 zu untersuchen.

Verzweige bei
"Bedingung erfüllt"



Verzweige bei
"Bedingung nicht erfüllt"



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		AUS	EQU	$8004
			ORG	$0190
0190	81		CMP A	#$40
	40			
0192	27		BEQ	M6
	0E			
0194	81		CMP A	#$20
	20			
0196	27		BEQ	M5
	05			
0198	86		LDA A	#$FF
	FF			
019A	7E		JMP	ANZ
	01			
	A4			
019D	86	M5	LDA A	#$06
	06			
019F	7E		JMP	ANZ
	01			
	A4			
01A2	86	M6	LDA A	#$05
	05			
01A4	B7	ANZ	STA A	AUS
	80			
	04			

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		AUS	EQU	$8004
			ORG	$0190
0190	81		CMP A	#$40
	40			
0192	26		BNE	W5
	05			
0194	86		LDA A	#$06
	06			
0196	7E		JMP	ANZ
	01			
	A4			
0199	81	W5	CMP A	#$20
	20			
019B	26		BNE	FEHL
	05			
019D	86		LDA A	#$05
	05			
019F	7E		JMP	ANZ
	01			
	A4			
01A2	86	FEHL	LDA A	#$FF
	FF			
01A4	B7	ANZ	STA A	AUS
	80			
	04			

5.1.3 Verzweigungen mit Vergleichsbefehlen des 8080/8085 **8080**

Die im Abschnitt 4.3.3. behandelten bedingten Sprungbefehle werten einzelne Bits des Bedingungsregisters aus. Besonders häufig werden das N-Bit (Negativ) und das Z-Bit (Zero =Null) abgefragt.

Die im Abschnitt 4.4.3. behandelten Vergleichsbefehle subtrahieren vom Inhalt des Akkumulators einen Vergleichswert und beeinflussen mit der Differenz das Bedingungsregister, ohne jedoch den Inhalt des Akkumulators zu verändern.

Transportbefehle (z.B. LDA, MOV, IN) beeinflussen nicht das Bedingungsregister. Vor bedingten Sprüngen muß daher der Akkumulator mit dem Befehl ADI 00H oder ORA A auf die Sprungbedingung geprüft werden.

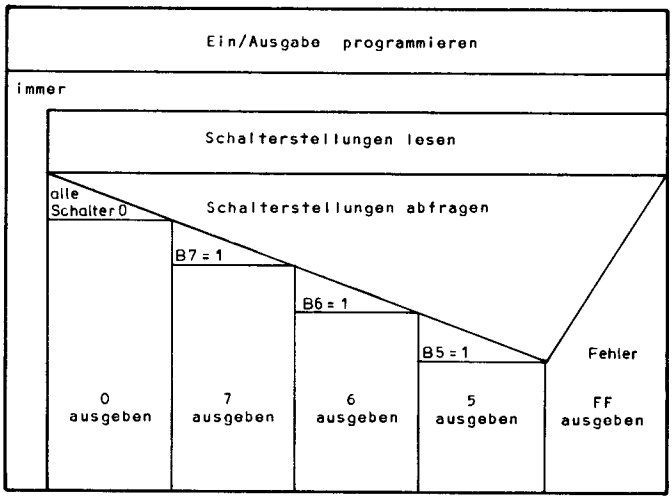
Beispiel:

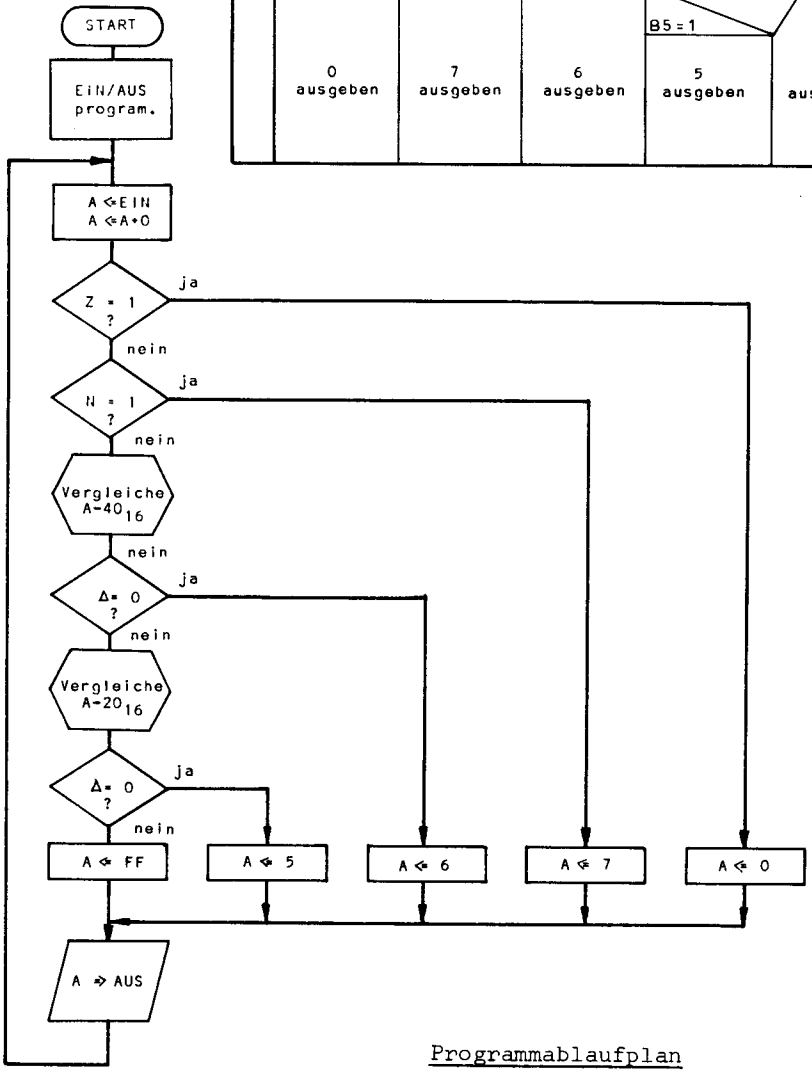
An einer Parallelschnittstelle liege eine Eingabeschaltung nach Bild 48 mit acht Kippschaltern, von denen nur die drei werthöchsten verwendet werden, die restlichen fünf seien ohne Bedeutung.

Für Schalter B7 = 1 werde die Zahl 7 dual angezeigt,
für Schalter B6 = 1 werde die Zahl 6 dual angezeigt und
für Schalter B5 = 1 werde die Zahl 5 dual angezeigt.
Sind alle Schalterstellungen Null, so sei die Ausgabe gelöscht.
Ist mehr als ein Schalter 1, so sollen als Fehlermeldung alle Anzeigen leuchten.

Die Aufgabe wird hier mit Programmverzweigungen gelöst, obwohl es auch andere Möglichkeiten gibt, wie z.B. das Suchen der Schalterstellungen in einer Tabelle.

Programmblockplan





Programmablaufplan

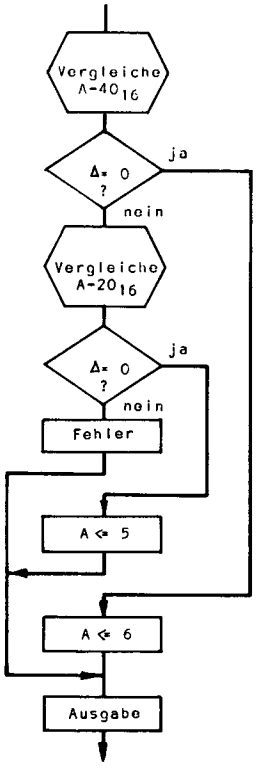
Programm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	D3		OUT	02H
	02			
2884	3E		MVI	A,00H
	00			
2886	D3		OUT	03H
	03			
2888	DB	SUS1	IN	01H
	01			
288A	C6		ADI	00H
	00			
288C	CA		JZ	M0
	A1			
	28			
288F	FA		JM	M7
	B0			
	28			
2892	FE		CPI	40H
	40			
2894	CA		JZ	M6
	AB			
	28			
289F	FE		CPI	20H
	20			
2899	CA		JZ	M5
	A6			
	28			
289C	3E		MVI	A,0FFH
	FF			
289E	C3		JMP	ANZ
	B2			
	28			
28A1	3E	M0	MVI	A,00H
	00			
28A3	C3		JMP	ANZ
	B2			
	28			
28A6	3E	M5	MVI	A,05H
	05			
28A8	C3		JMP	ANZ
	B2			
	28			
28AB	3E	M6	MVI	A,06H
	06			
28AD	C3		JMP	ANZ
	B2			
	28			
28B0	3E	M7	MVI	A,07H
	07			
28B2	D3	ANZ	OUT	00H
	00			
28B4	C3		JMP	SUS1
	88			
	28			
			END	

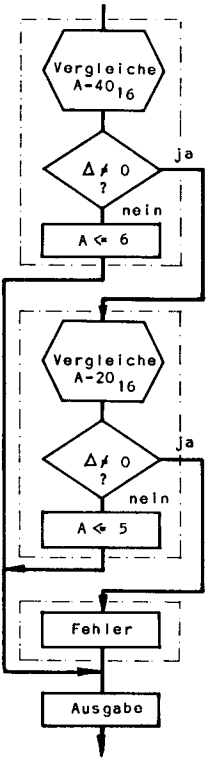
Bei den Vergleichen des vorigen Beispiels lagen die Abfrage und die Behandlung eines Falls im Programm weit voneinander entfernt. Dies lag an dem Befehl "Springe bei Null", der nach der Bejahung der Sprungbedingung in ein weit entferntes Programmstück verzweigte. Diese Technik "Springe, wenn die Bedingung erfüllt ist" ist zwar logisch einfach, macht das Programm jedoch unübersichtlich.

Behandelt man jedoch einen Fall sofort nach seiner Abfrage, so muß man die Sprungbedingung umdrehen, also bei "Nicht erfüllt" zu weiteren Abfragen springen. Das folgende Beispiel vergleicht beide Verfahren. Wie vorher sind die Fälle Akku = 40H und Akku = 20H zu untersuchen.

Springe bei
"Bedingung erfüllt"



Springe bei
"Bedingung nicht erfüllt"



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2890H
2890	FE		CPI	40H
	40			
2892	CA		JZ	M6
	A4			
	28			
2895	FE		CPI	20H
	20			
2897	CA		JZ	M5
	9F			
	28			
289A	3E		MVI	A,0FFH
	FF			
289C	C3		JMP	ANZ
	A6			
	28			
289F	3E	M5	MVI	A,05H
	05			
28A1	C3		JMP	ANZ
	A6			
	28			
28A4	3E	M6	MVI	A,06H
	06			
28A6	D3	ANZ	OUT	00H
	00			

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2890H
2890	FE		CPI	40H
	40			
2892	C2		JNZ	W5
	9A			
	28			
2895	3E		MVI	A,06H
	06			
2897	C3		JMP	ANZ
	A6			
	28			
289A	FE	W5	CPI	20H
	20			
289C	C2		JNZ	FEHL
	A4			
	28			
289F	3E		MVI	A,05H
	05			
28A1	C3		JMP	ANZ
	A6			
	28			
28A4	3E	FEHL	MVI	A,0FFH
	FF			
28A6	D3	ANZ	OUT	00H
	00			

5.2 Verzweigungen mit Hilfe von logischen Befehlen

Oft entscheidet nur ein bestimmtes Bit innerhalb eines Bytes über eine Programmverzweigung, die restlichen sieben Bits können beliebig gesetzt sein, sind aber ohne Bedeutung. Nur das werthöchste Bit (Vorzeichen) ist ohne Berücksichtigung der restlichen Bits durch die Befehle "Springe bei positiv" und"Springe bei negativ" abfragbar.
Zum Ausblenden bestimmter Bitpositionen dient eine "Maske", die über die UND-Funktion mit dem zu untersuchenden Byte verknüpft wird. Das Ergebnis (logische Produkt) ist in den Bitpositionen 0, in denen die Maske 0 ist und behält überall dort seinen alten Wert, wo die Maske 1 ist.

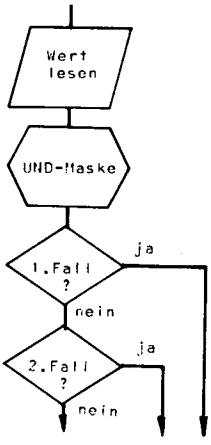
Beispiel:

Bit 0, das wertniedrigste Bit des Akkumulators sei ohne Berücksichtigung der restlichen Bits zu untersuchen.

Akkumulator	1010101X	X beliebig 0 oder 1
Maske	00000001	
Ergebnis	0000000X	

Durch die Maske 00000001 wurden alle unwichtigen Bits 0 gesetzt, nur die gewünschte Bitposition behielt ihren alten Wert.

Werden durch eine Maske einzelne Bitpositionen herausgegriffen, so müssen die restlichen Verzweigungen durch Vergleichs- und Sprungbefehle vorgenommen werden.



Soll umgekehrt in eine bestimmte Bitposition ein 1er Bit eingefügt werden, so ist der Speicherstelle mit Hilfe der ODER-Funktion eine Konstante zu überlagern, die an den gewünschten Bitpositionen 1er Bits enthält.

Beispiel:

Bit 7, das werthöchste Bit, des Akkumulators werde immer 1 gesetzt, die restlichen Bitpositionen sollen unverändert bleiben.

Akkumulator	X1010101	X beliebig 0 oder 1
Konstante	10000000	
Ergebnis	11010101	

Durch die Konstante 10000000 wurde eine 1 in die Bitposition 7 gesetzt, die übrigen Bits behielten ihren alten Wert.

Das später noch zu behandelnde exklusive ODER (XOR, EOR)komplementiert (aus 0 mach 1, aus 1 mach 0) den Inhalt eines Registers an den Stellen, an denen eine Maske 1er Bits enthält. Damit kann die Logik von Eingabeschaltern umgedreht werden.

Beispiel:

Die Bitpositionen 0, 1 und 2 des Akkumulators seien zu komplementieren, der Rest des Akkumulators bleibe unverändert.

Akkumulator	10101010
Maske	00000111
Ergebnis	10101101

Durch die Maske 00000111 wurden die letzten drei Bitpositionen komplementiert (1er Komplement), der Rest bleibt unberührt. Der Abschnitt 6.1. bringt weitere Anwendungen der logischen Befehle.

5.2.1 Verzweigungen mit logischen Befehlen des KIM-1 KIM

Die beiden folgenden logischen Befehle verknüpfen den Inhalt des Akkumulators mit einer Konstanten, die im 2. Byte des Befehls steht. Die anderen logischen Funktionen und die anderen Adressierungsarten folgen im Abschnitt 6.1.1.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	V	B	D	I	Z	C	
AND	# konst	unmittelbar	29	2	2	x					x		Akku ← Akku AND Konstante
ORA	# konst	unmittelbar	09	2	2	x					x		Akku ← Akku OR Konstante

AND bedeutet UND; ORA bedeutet ODER.

Beispiel:

Ähnlich wie beim Beispiel Abschnitt 5.1.1. sollen die Bitpositionen 0 bis 4 einer Eingabe ohne Bedeutung sein und durch eine Maske 11100000 = E0 ausgeblendet werden.

```
  Hexadezimal          Symbolisch
Adresse Inhalt   Name   Befehl    Operand

                 EIN    =$1702    Datenreg. B

                        *=$0310
0310     AD             LDA       EIN
         02
         17
0313     29             AND       #$E0
         E0
```

Beispiel:

Unabhängig von ihrem eingelesenen Wert sollen die Bitpositionen 4 bis 7 für die Ausgabe auf 1 gesetzt werden, die vier wertniedrigsten Bitpositionen bleiben erhalten. Die Konstante lautet 11110000 = F0.

```
Hexadezimal                Symbolisch
Adresse Inhalt   Name      Befehl    Operand

                 AUS       =$1700
                 EIN       =$1702

                           *=$0310
0310     AD                LDA       EIN
         02
         17
0313     09                ORA       #$F0
         F0
0315     8D                STA       AUS
         00
         17
```

5.2.2 Verzweigungen mit logischen Befehlen des 6800 **6800**

Die folgenden logischen Befehle verknüpfen den Inhalt eines Akkumulators mit einer Konstanten, die im zweiten Byte des Befehls steht. Die anderen logischen Funktionen und die anderen Adressierungsarten folgen im Abschnitt 6.1.2.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen -- -- H I N Z V C	
AND A	# konst	unmittelbar	84	2	2	X X 0	Akku A ← AKKU A UND Konstante
AND B	# konst	unmittelbar	C4	2	2	X X 0	Akku B ← Akku B UND Konstante
ORA A	# konst	unmittelbar	8A	2	2	X X 0	Akku A ← Akku A ODER Konstante
ORA B	# konst	unmittelbar	CA	2	2	X X 0	Akku B ← Akku B ODER Konstante

AND bedeutet UND, ORA bedeutet ODER.

Beispiel:

Ähnlich wie im Abschnitt 5.1.2. sollen die Bitpositionen 0 bis 4 einer Eingabe ohne Bedeutung sein und durch die Maske 11100000 = E0 ausgeblendet werden.

```
Hexadezimal                Symbolisch
Adresse Inhalt   Name      Befehl    Operand

                 EIN       EQU       $8006

                           ORG       $0190
0190     B6                LDA A     EIN
         80
         06
0913     84                AND A     #$E0
         E0
```

Beispiel:

Unabhängig von ihrem eingelesenen Wert sollen die vier werthöchsten Bitpositionen 4 bis 7 für die Ausgabe auf 1 gesetzt werden, die vier wertniedrigsten Bitpositionen bleiben erhalten. Die Konstante lautet 11110000 = F0.

```
Hexadezimal                Symbolisch
Adresse Inhalt   Name      Befehl    Operand

                 AUS       EQU       $8004
                 EIN       EQU       $8006

                           ORG       $0190
0190     B6                LDA A     EIN
         80
         06
0193     8A                ORA A     #$F0
         F0
0195     B7                STA A     AUS
         80
         04
```

5.2.3 Verzweigungen mit logischen Befehlen des 8080/8085

Die beiden folgenden logischen Befehle verknüpfen den Inhalt des Akkumulators mit einer Konstanten, die im zweiten Byte des Befehls steht. Die anderen logischen Funktionen und die anderen Adressierungsarten folgen im Abschnitt 6.1.3.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N	Z	H	P	C	
ANI	konst	unmittelbar	E6	2	7	X	X	X	X	X	Akku ← Akku UND Konstante
ORI	konst	unmittelbar	F6	2	7	X	X	X	X	X	Akku ← Akku ODER Konstante

ANI bedeutet AND Immediate = logisches UND mit der unmittelbar folgenden Konstanten

ORI bedeutet OR Immediate = logisches ODER mit der unmittelbar folgenden Konstanten.

Beispiel:

Ähnlich wie im Abschnitt 5.1.3. sollen die Bitpositionen 0 bis 4 einer Eingabe ohne Bedeutung sein und durch die Maske 11100000 = E0 ausgeblendet werden.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2880H
2880    DB             IN       01H
        01
2882    E6             ANI      0E0H
        E0
```

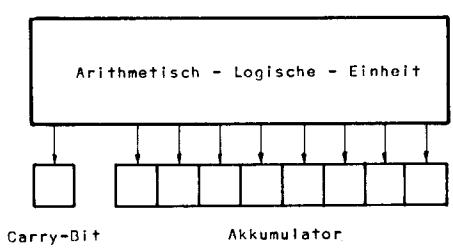
Beispiel:

Unabhängig von ihrem eingelesenen Wert sollen die vier werthöchsten Bitpositionen 4 bis 7 für die Ausgabe auf 1 gesetzt werden, die wertniedrigsten Bitpositionen 0 bis 3 bleiben erhalten. Die Konstante lautet 11110000 = F0.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2880H
2880    DB             IN       01H
        01
2882    F6             ORI      0F0H
        F0
2884    D3             OUT      00H
        00
```

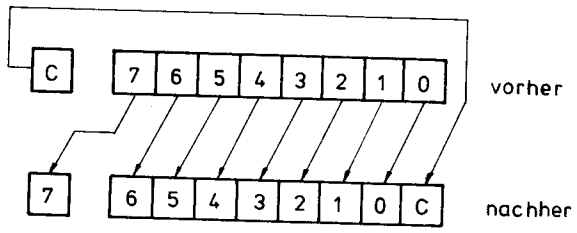
5.3 Verzweigungen mit Hilfe von Schiebebefehlen A

Das Carry-Bit ist eine Verlängerung des Akkumulators; es speichert bei der Addition positiver Dualzahlen den Übertrag der werthöchsten Stelle.



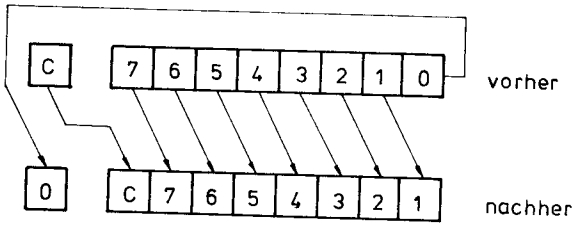
Mit Hilfe von Schiebebefehlen, die den Inhalt des Akkumulators verschieben und zusätzlich ein Bit des Akkumulators in das Carry-Bit bringen, können die einzelnen Bits des Akkumulators getrennt als Sprungbedingung verwendet werden. Die bedingten Sprünge werten das C-Bit und das N-Bit des Bedingungsregisters aus.

Der Befehl "Schiebe zyklisch links" verschiebt die neun Bits Carry und Akkumulator um ein Bit nach links. Bitposition 7 des Akkumulators geht in das Carry-Bit, das Carry-Bit geht nach Bitposition 0 des Akkumulators.



Schiebe 1 Bit zyklisch links

Der Befehl "Schiebe zyklisch rechts" verschiebt die neun Bits Carry und Akkumulator um ein Bit nach rechts. Bitposition 0 des Akkumulators geht in das Carry-Bit, das Carry-Bit geht nach Bitposition 7.



Schiebe 1 Bit zyklisch rechts

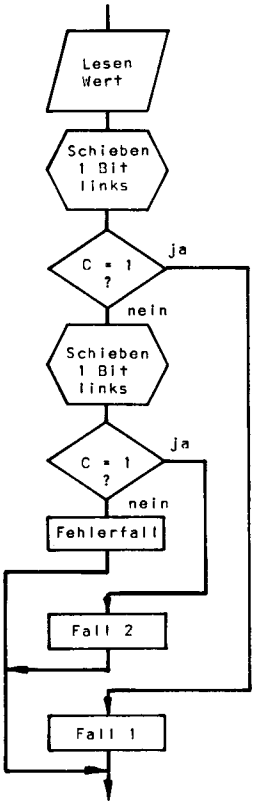
Für die Verzweigungen dienen nun die Befehle

Springe bei C = 1
Springe bei C = 0
Springe bei N = 1
Springe bei N = 0

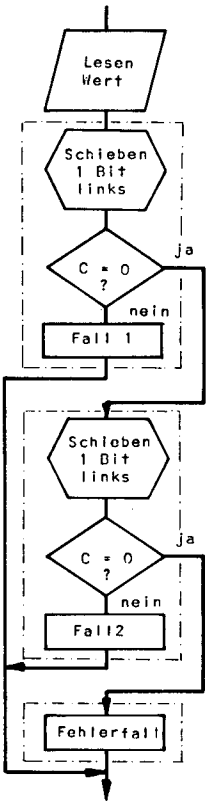
Die Schieberichtung entscheidet, welche Bits vorrangig behandelt werden. Der Akkumulatorinhalt wird durch das Schieben verändert; nach neun Schiebeoperationen ist der ursprüngliche Zustand wiederhergestellt.

Das folgende Beispiel zeigt wieder die beiden Möglichkeiten, entweder nach der Abfrage zur weiteren Behandlung des Falls zu verzweigen oder besser den Fall nach der Abfrage sofort zu behandeln.

Springe bei
"Bedingung erfüllt"



Springe bei
"Bedingung nicht erfüllt"

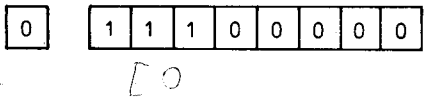


Eine besondere Anwendung finden Schiebebefehle bei Programmverzweigungen, die sich zyklisch wiederholen.

Beispiel:

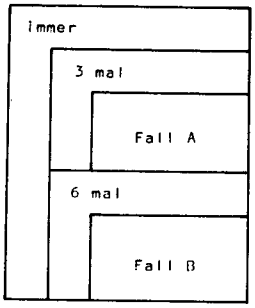
In einer unendlichen Schleife werde dreimal das Programmstück A und sechsmal das Programmstück B durchlaufen.

Dazu werden der Akkumulator und das C-Bit mit folgender "Sprungleiste" geladen:

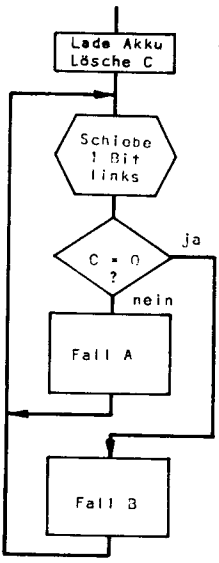


Schiebt man nun in einer Schleife das Register und das C-Bit zyklisch nach links und fragt das C-Bit ab, so ist es dreimal 1 und sechsmal 0. Durch Kombination und Schachtelung mehrerer Sprungleisten lassen sich auch andere Teilerverhältnisse erreichen.

Programmblockplan



Programmablaufplan



5.3.1 Verzweigungen mit Schiebebefehlen des KIM-1 KIM

Die beiden folgenden Schiebebefehle verschieben den Akkumulator mit dem Carry-Bit um ein Bit zyklisch nach rechts oder links. Der Abschnitt 6.2.1. zeigt weitere Schiebebefehle und Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
ROLA		code-eigen	2A	1	2	X				X	X	Schiebe C-Bit und Akku 1 Bit zykl. links
RORA		code-eigen	6A	1	2	X				X	X	Schiebe C-Bit und Akku 1 Bit zykl. rechts

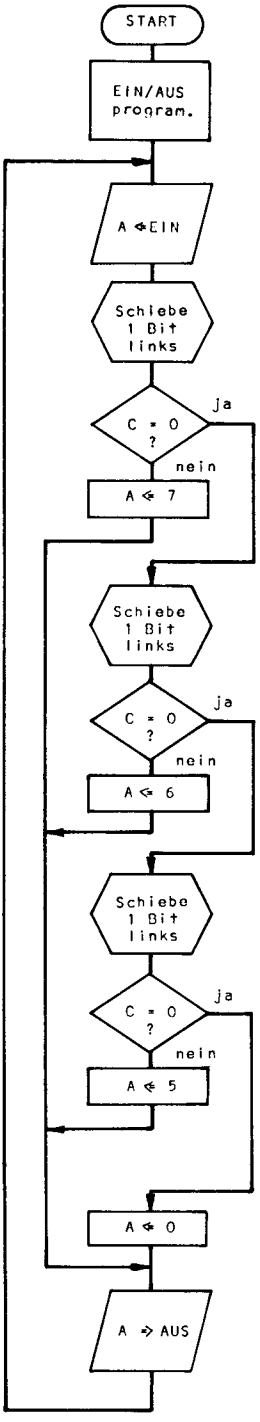
ROL bedeutet ROtate Left = schiebe zyklisch links.
ROR bedeutet ROtate Right = schiebe zyklisch rechts.

Beispiel:

Es werde wieder wie in den vorigen Abschnitten eine Eingabe von acht Kippschaltern untersucht.

Für Schalter B7 = 1 werde die Zahl 7 dual angezeigt,
für Schalter B6 = 1 werde die Zahl 6 dual angezeigt und
für Schalter B5 = 1 werde die Zahl 5 dual angezeigt.

Die höherwertigen Bits haben Vorrang vor den niederwertigen, einen Fehlerfall gibt es nicht mehr. In dem Programm wird nur das C-Bit ausgewertet.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
		AUS	=$1700	
		RAU	=$1701	
		EIN	=$1702	
		REI	=$1703	
			*=$0300	
0300	A9	START	LDA	#$00
	00			
0302	8D		STA	REI
	03			
	17			
0305	A9		LDA	#$FF
	FF			
0307	8D		STA	RAU
	01			
	17			
030A	AD	SUSI	LDA	EIN
	02			
	17			
030D	2A		ROLA	
030E	90		BCC	M6
	05			
0310	A9		LDA	#$07
	07			
0312	4C		JMP	ANZ
	27			
	03			
0315	2A	M6	ROLA	
0316	90		BCC	M5
	05			
0318	A9		LDA	#$06
	06			
031A	4C		JMP	ANZ
	27			
	03			
031D	2A	M5	ROLA	
031E	90		BCC	NULL
	05			
0320	A9		LDA	#$05
	05			
0322	4C		JMP	ANZ
	27			
	03			
0325	A9	NULL	LDA	#$00
	00			
0327	8D	ANZ	STA	AUS
	00			
	17			
032A	4C		JMP	SUSI
	0A			
	03			
			.END	

5.3.2 Verzweigungen mit Schiebebefehlen des 6800

Die vier folgenden Schiebebefehle verschieben jeweils einen der beiden Akkumulatoren mit dem Carry-Bit zyklisch um ein Bit nach links oder rechts. Das V-Bit (OVerflow = Überlauf) ist das exklusive ODER des N- und C-Bits nach der Verschiebung.

N	C	V
0	0	0
0	1	1
1	0	1
1	1	0

Der Abschnitt 6.2.2. zeigt weitere Schiebebefehle und Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen -HI NZVC	
ROL A		code-eigen	49	1	2	xxxx	Schiebe C-Bit und Akku A 1 Bit zykl. links
ROL B		code-eigen	59	1	2	xxxx	Schiebe C-Bit und Akku B 1 Bit zykl. links
ROR A		code-eigen	46	1	2	xxxx	Schiebe C-Bit und Akku A 1 Bit zykl. rechts
ROR B		code-eigen	56	1	2	xxxx	Schiebe C-Bit und Akku B 1 Bit zykl. rechts

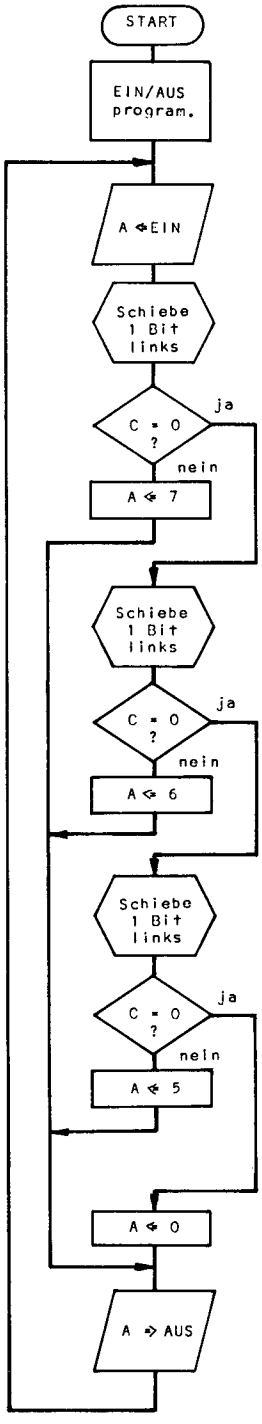
ROL bedeutet ROtate Left = verschiebe zyklisch links.
ROR bedeutet ROtate Right = verschiebe zyklisch rechts.

Beispiel:

Es werde wieder wie in den vorigen Abschnitten eine Eingabe von acht Kippschaltern untersucht.

Für Schalter B7 = 1 werde die Zahl 7 dual angezeigt,
für Schalter B6 = 1 werde die Zahl 6 dual angezeigt und
für Schalter B5 = 1 werde die Zahl 5 angezeigt.

Die höherwertigen Bits haben Vorrang vor den niederwertigen, einen Fehlerfall gibt es nicht mehr. In dem Programm wird nur das C-Bit ausgewertet.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	EQU	$8004
		KONA	EQU	$8005
		EIN	EQU	$8006
		KONB	EQU	$8007
			ORG	$0180
0180	73	START	COM	AUS
	80			
	04			
0183	86		LDA A	#$04
	04			
0185	B7		STA A	KONA
	80			
	05			
0188	B7		STA A	KONB
	80			
	07			
018B	B6	SUSI	LDA A	EIN
	80			
	06			
018E	49		ROL A	
018F	24		BCC	M6
	05			
0191	86		LDA A	#$07
	07			
0193	7E		JMP	ANZ
	01			
	A7			
0196	49	M6	ROL A	
0197	24		BCC	M5
	05			
0199	86		LDA A	#$06
	06			
019B	7E		JMP	ANZ
	01			
	A7			
019E	49	M5	ROL A	
019F	24		BCC	NULL
	05			
01A1	86		LDA A	#$05
	05			
01A3	7E		JMP	ANZ
	01			
	A7			
01A6	4F	NULL	CLR A	
01A7	B7	ANZ	STA A	AUS
	80			
	04			
01AA	7E		JMP	SUSI
	01			
	8B			
			END	

5.3.3 Verzweigungen mit Schiebebefehlen des 8080/8085

Die beiden folgenden Schiebebefehle verschieben den Akkumulator mit dem Carry-Bit um ein Bit zyklisch nach rechts oder links. Der Abschnitt 6.2.3. zeigt weitere Schiebebefehle.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ H P C	
RAL		code-eigen	17	1	4	X	Schiebe C-Bit und Akku 1 Bit zykl. links
RAR		code-eigen	1F	1	4	X	Schiebe C-Bit und Akku 1 Bit zykl. rechts

RAL bedeutet Rotate Akkumulator Left = verschiebe Akkumulator zyklisch links.

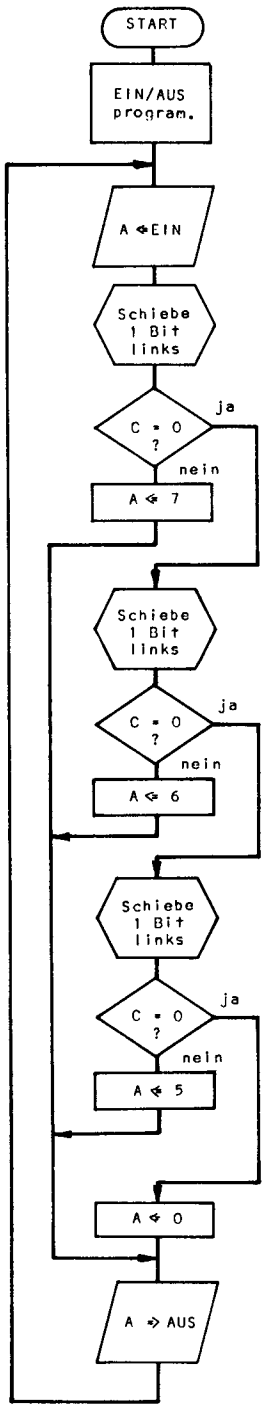
RAR bedeutet Rotate Akkumulator Right = verschiebe Akkumulator zyklisch rechts.

Beispiel:

Es werde wieder wie in den vorigen Abschnitten eine Eingabe von acht Kippschaltern untersucht.

Für Schalter B7 = 1 werde die Zahl 7 dual angezeigt,
für Schalter B6 = 1 werde die Zahl 6 dual angezeigt und
für Schalter B5 = 1 werde die Zahl 5 dual angezeigt.

Die höherwertigen Bits haben Vorrang vor den niederwertigen, einen Fehlerfall gibt es nicht mehr. Im Programm wird nur das C-Bit ausgewertet.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	D3		OUT	02H
	02			
2884	3E		MVI	A,00H
	00			
2886	D3		OUT	03H
	03			
2888	DB	SUSI	IN	01H
	01			
288A	17		RAL	
288B	D2		JNC	M6
	93			
	28			
288E	3E		MVI	A,07H
	07			
2890	C3		JMP	ANZ
	A7			
	28			
2893	17	M6	RAL	
2894	D2		JNC	M5
	9C			
	28			
2897	3E		MVI	A,06H
	06			
2899	C3		JMP	ANZ
	A7			
	28			
289C	17	M5	RAL	
289D	D2		JNC	NULL
	A5			
	28			
28A0	3E		MVI	A,05H
	05			
28A2	C3		JMP	ANZ
	A7			
	28			
28A5	3E	NULL	MVI	A,00H
	00			
28A7	D3	ANZ	OUT	00H
	00			
28A9	C3		JMP	SUSI
	88			
	28			
			END	

5.4 Der berechnete (indirekte) Sprung

Bei den bisher behandelten Programmverzweigungen wurde das Sprungziel fest im Befehl vorgegeben. Dies kann bei weitverzweigten Entscheidungen zu umfangreichen Programmen mit langer Bearbeitungszeit führen. Sollen z.B. für alle 256 möglichen Bitkombinationen eines Bytes Verzweigungen programmiert werden, so wären 255 Vergleiche erforderlich.

Ähnlich wie bei der indirekten Adressierung von Daten ist es auch bei Befehlen möglich, eine Sprungadresse aus einem Register oder einer Datenspeicherstelle in das Befehlszählregister zu bringen und damit einen unbedingten Sprung auszulösen.
Bild 102 zeigt einen indirekten Sprung. Die indirekte Adresse 1111 weist auf eine Speicherstelle hin, die das eigentliche Sprungziel 3333 enthält; das Sprungziel kann damit eingegeben oder berechnet werden.

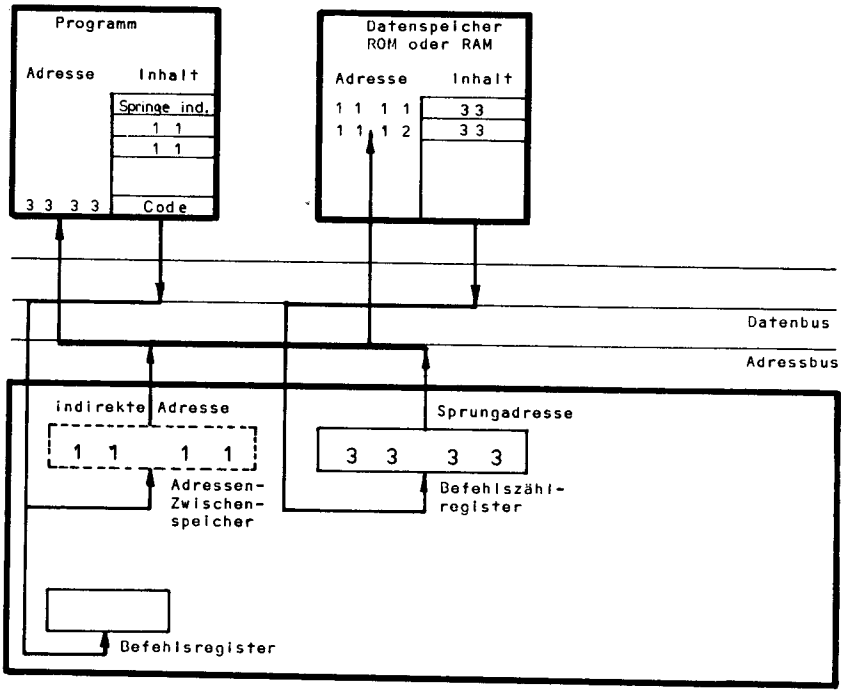


Bild 102: Indirekter Sprung

Der in Bild 102 dargestellte indirekte Sprung wird in folgenden Schritten ausgeführt:

1. Funktionscode des indirekten Sprungbefehls holen.
2. Byte 2 und 3 mit der indirekten Sprungadresse holen.
3. Indirekte Adresse aussenden.
4. Richtige Sprungadresse in das Befehlszählregister holen.
5. Richtige Sprungadresse aussenden.
6. Funktionscode des Sprungziels holen.

Bei der Programmierung eines indirekten Sprunges muß ein Zusammenhang zwischen der entscheidenden Größe (z.B. eingelesene Schalterstellungen) und der Adresse des Sprungziels hergestellt werden. Dies sollte bereits bei der Wahl der Eingabeleitungen berücksichtigt werden. Weiter ist zu bedenken, daß die richtige Sprungadresse aus 16 Bits besteht und daß zwischen den Sprungzielen Platz für die entsprechenden Programmteile vorhanden sein muß.

a. Der Eingabewert ist Bestandteil der Sprungadresse

Das einfachste und schnellste Verfahren ermittelt die Sprungadresse aus einem Eingabewert durch logische und arithmetische Operationen oder Verschiebungen.

Beispiel:

Die zwei wertniedrigsten Bits einer Eingabe ergeben $2^2 = 4$ verschiedene Sprungziele, die sechs werthöheren Bits seien ohne Bedeutung.

Eingabe 00_2 Sprung nach Adresse 0010
Eingabe 01_2 Sprung nach Adresse 0014
Eingabe 10_2 Sprung nach Adresse 0018
Eingabe 11_2 Sprung nach Adresse 001C

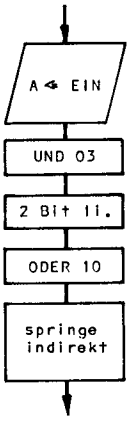
Die Umrechnungsformel lautet:

Adresse = (Eingabewert X 4) + 10_{16}

Man beachte, daß das Schieben um ein Bit nach links einer Multiplikation mit 2 entspricht.

Eingabe	XXXXXXXX
UND-Maske	00000011
Ergebnis	000000XX
zweimal links schieben	0000XX00
ODER-Konstante	00010000
Sprungadresse	0001XX00

XX = 00 : Adresse 10
XX = 01 : Adresse 14
XX = 10 : Adresse 18
XX = 11 : Adresse 1C



Bei schwierigen Fallunterscheidungen läßt sich oft keine einfache und zugleich eindeutige Zuordnung zwischen Eingabewert und Sprungadresse finden oder es ist nicht möglich, zwischen den Sprungzielen genügend Platz für die Programmteile zu finden oder es entstehen zu große Lücken zwischen den Sprungzielen. Bei nicht eindeutiger Zuordnung kann man zunächst mehrere Fälle für ein Sprungziel zulassen (Vorentscheidung) und dann im angesprungenen Programmteil durch Vergleichs- und Sprungbefehle die endgültige Programmverzweigung vornehmen.

b. Der Eingabewert ist Adresse einer Tabelle von Sprungzielen

Eine Tabelle enthält die Adressen der verschiedenen Sprungziele. Im einfachsten Fall wird nur das wertniedrigste Byte der Sprungadresse gespeichert, das werthöhere ist oft konstant. Es ist aber auch möglich, beide Teile der Sprungadresse in zwei aufeinanderfolgenden Bytes der Tabelle abzulegen.

Beispiel:

Wie im Beispiel Punkt a. gelte folgende Zuordnung:

Eingabe $00_2 = 00_{16}$ Sprung nach Adresse 10_{16}
Eingabe $01_2 = 01_{16}$ Sprung nach Adresse 14_{16}
Eingabe $10_2 = 02_{16}$ Sprung nach Adresse 18_{16}
Eingabe $11_2 = 03_{16}$ Sprung nach Adresse $1C_{16}$

Das höherwertige Byte der Adresse sei konstant 00.

Die Sprungtabelle lautet hexadezimal:

Adresse	Inhalt
0000	10
0001	14
0002	18
0003	1C

Der Eingabewert wird zunächst in eine Datenadresse umgeformt, die dann mit Hilfe eines Indexregisters das Ziel des indirekten Sprunges auswählt. Enthält die Tabelle die volle 2-Byte-Sprungadresse, so ist bei der Bildung der Tabellenadresse zu berücksichtigen, daß nur jeder zweite Tabellenwert adressiert wird. Diese Methode hat den Vorteil, daß die anzuspringenden Programmteile beliebig lang sein können.

c. Der Eingabewert wird in einer Tabelle gesucht.

Bei sehr schwierigen Zuordnungen kann es vorteilhafter sein, den Eingabewert und die Sprungadresse in einer Tabelle zu speichern und dann die Tabelle linear oder binär nach dem Eingabewert zu durchsuchen. Der gefundene Wert zeigt dann auf die Sprungadresse.

Beispiel:

Wie im Punkt a. gelte folgende Zuordnung:

Eingabe $00_2 = 00_{16}$ Sprung nach 10_{16}
Eingabe $01_2 = 01_{16}$ Sprung nach 14_{16}
Eingabe $10_2 = 02_{16}$ Sprung nach 18_{16}
Eingabe $11_2 = 03_{16}$ Sprung nach $1C_{16}$

Das höherwertige Byte der Adresse sei wieder konstant 00.

Sprungtabelle hexadezimal:

Adresse	Inhalt	Bedeutung
4711	00	Eingabe
4712	10	Sprungadresse
4713	01	Eingabe
4714	14	Sprungadresse
4715	02	Eingabe
4716	18	Sprungadresse
4717	03	Eingabe
4718	1C	Sprungadresse

Die Tabelle muß nun in einer Indexschleife mit der Schrittweite zwei nach dem Eingabewert durchsucht werden.

5.4.1 Der indirekte Sprung des KIM-1 KIM

Der indirekte Sprung ist beim KIM-1 als Befehl verfügbar.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	
JMP	(Adresse)	indirekt	6C	3	5		Springe indirekt, Sprungziel in Speicherst.

Die im Befehl stehende indirekte Adresse besteht aus zwei Bytes und erfordert damit absolute Adressierung.

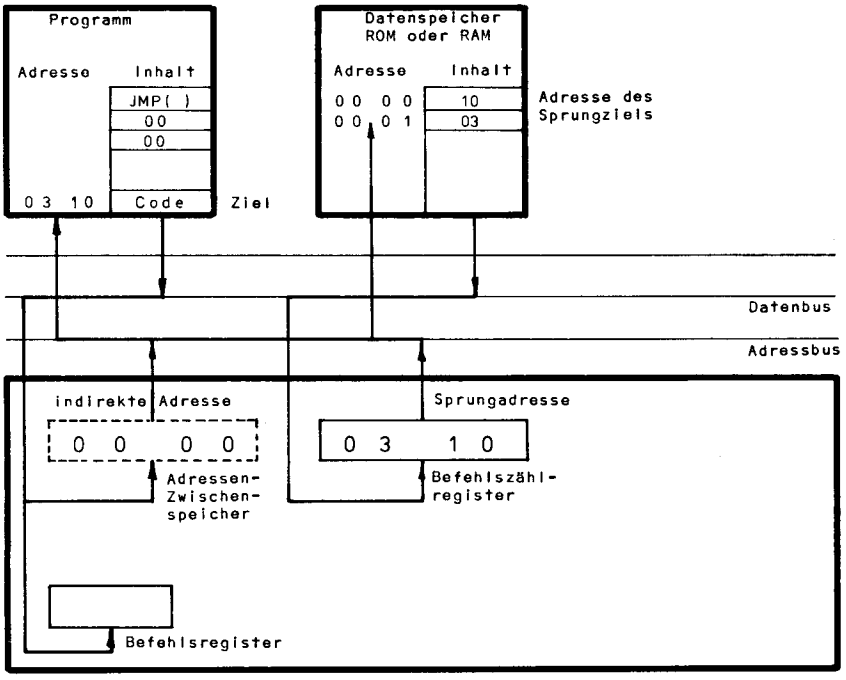


Bild 103: Indirekter Sprung des KIM-1

Bild 103 zeigt die Wirkung des Befehls. Die Speicherstelle 0000 enthalte das wertniedrigere Byte, die Speicherstelle 0001 das werthöhere Byte der Sprungadresse 0310. Im Befehl "Springe indirekt" steht die indirekte Adresse 0000.

Das in Bild 103 dargestellte Beispiel wird in folgenden Schritten ausgeführt:

1. Funktionscode des indirekten Sprungbefehls in das Befehlsregister bringen
2. Niederwertiges Byte 00 der indirekten Adresse holen
3. Höherwertiges Byte 00 der indirekten Adresse holen
4. Indirekte Adresse 0000 aussenden
5. Niederwertiges Byte der 10 der Sprungadresse holen
6. Um 1 erhöhte indirekte Adresse 0001 aussenden
7. Höherwertiges Byte 03 der Sprungadresse holen
8. Sprungadresse 0310 in den Befehlszähler bringen
9. Sprungadresse 0310 aussenden
10. Funktionscode des Sprungziels in das Befehlsregister holen.

Beispiel:

Es soll indirekt gesprungen werden. Eine Speicherstelle KATZE enthalte die Adresse des Sprungziels MAUS.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0000	
0000	10	KATZE	.WORD	MAUS
0001	03			
			*=$0300	
0300	6C	START	JMP	(KATZE)
	00			
	00			
0310	A9	MAUS	LDA	#$FF
	FF			

5.4.2 Der indirekte Sprung mit dem 6800 **6800**

Der Mikroprozessor 6800 hat keinen eingebauten indirekten Sprungbefehl. Der unbedingte Sprung JMP kann jedoch mit indizierter Adressierung verwendet werden. Das X-Register enthält die Sprungadresse, im zweiten Byte steht eine zusätzliche Sprungweite oder Abstand, die dazuaddiert wird.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI NZVC	
JMP	konst,X	indiziert	6E	2	4		Springe indiziert,Sprungziel=X-Reg.+konst

Das X bedeutet, daß zu der im Befehl stehenden Konstanten noch der Inhalt des X-Registers addiert wird.

Für den indirekten Sprung sind beim 6800 zwei Befehle erforderlich:

1. Laden des Indexregisters mit der Sprungadresse
2. Indizierter Sprung

Bild 104 zeigt, wie zunächst das Indexregister mit dem Inhalt der Speicherstellen 0000 und 0001 geladen wird. Dies ist die Sprungadresse 0190. Der anschließende indizierte Sprung hat den Abstand 00 im zweiten Byte.

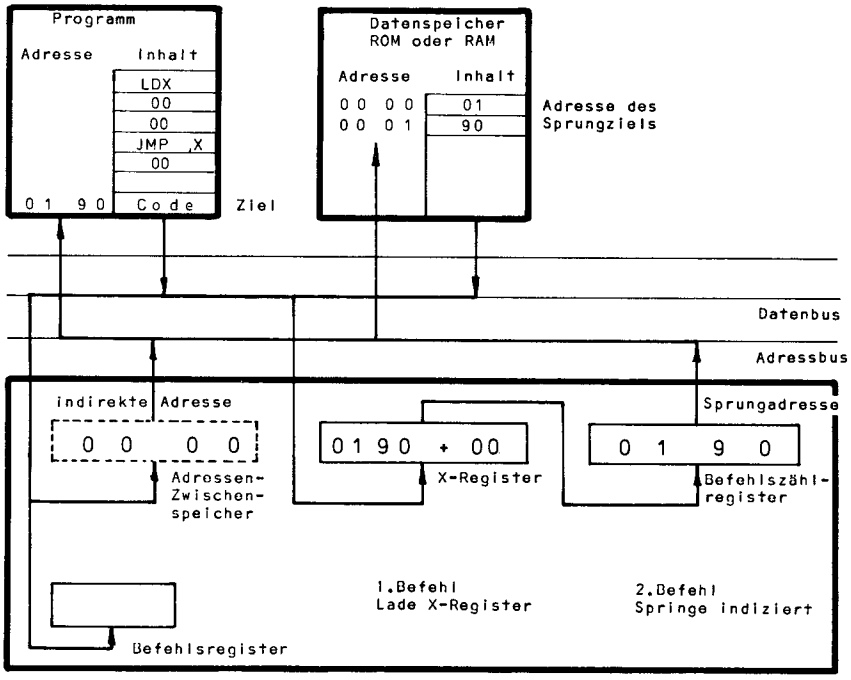


Bild 104: Indirekter Sprung mit dem 6800

Beispiel:

Es soll indirekt gesprungen werden. Eine Speicherstelle KATZE enthalte die Adresse des Sprungziels MAUS.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000    01     KATZE   FDB      MAUS
0001    90

                       ORG      $0180
0180    FE     START   LDX      KATZE
        00
        00
0183    6E             JMP      00,X
        00
 |      |
 |      |
 |      |
0190    86     MAUS    LDA A    #$FF
        FF
```

5.4.3 Der indirekte Sprung mit dem 8080/8085

Die Mikroprozessoren 8080 und 8085 haben keinen eingebauten indirekten Sprungbefehl. Der Befehl PCHL überträgt jedoch den Inhalt des Registerpaares H,L in das Befehlszählregister.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H -P-C	
PCHL		code-eigen	E9	1	5		Lade Befehlszählregister mit H,L-Register

PCHL bedeutet "Lade den Program-Counter (Befehlszähler) mit dem H,L -Register.

Für den indirekten Sprung sind beim 8080/8085 zwei Befehle erforderlich:

1. Laden des Registerpaares H,L mit der Sprungadresse
2. Laden des Befehlszählers mit dem Inhalt des Registerpaares H,L

Bild 105 zeigt, wie zunächst das Registerpaar H,L mit dem Inhalt der Speicherstellen 2000 und 2001, der Sprungadresse 2890, geladen wird. Der nächste Befehl überträgt diese Adresse in das Befehlszählregister.

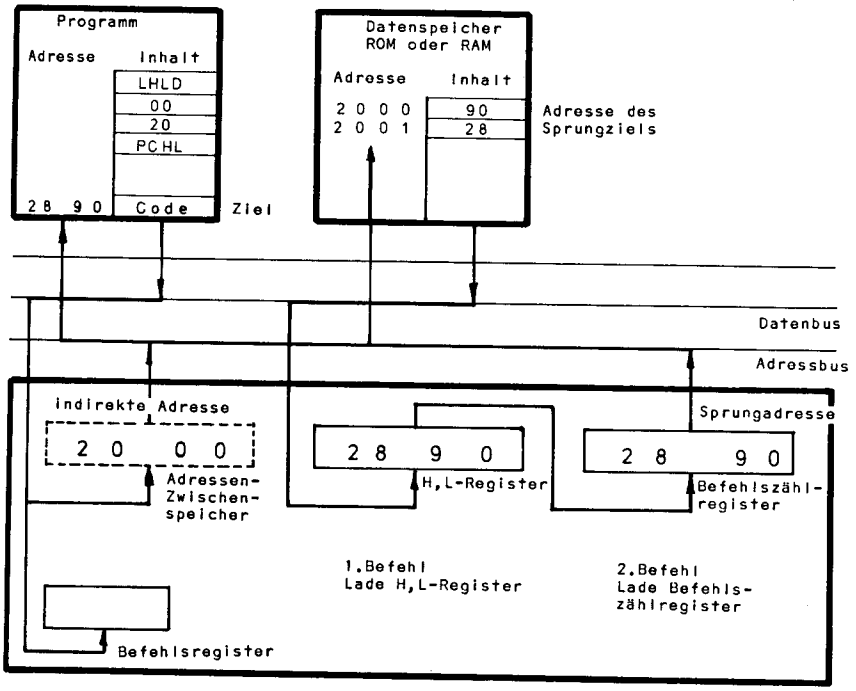


Bild 105: Indirekter Sprung mit dem 8080/8085

Beispiel:

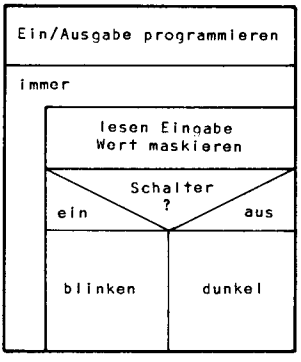
Es soll indirekt gesprungen werden. Eine Speicherstelle KATZE enthalte die Adresse des Sprungziels MAUS.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000	90	KATZE	DW	MAUS
	28			
			ORG	2880H
2880	2A		LHLD	KATZE
	00			
	20			
2883	E9		PCHL	
	⋮			
2890	3E	MAUS	MVI	A,OFFH
	FF			

Übungen zum Abschnitt 5. **A**

1.Aufgabe

An einer Parallelschnittstelle liegt eine Schaltung nach Bild 48 mit acht Kippschaltern und acht Anzeigeeinheiten. Die vier höherwertigen Schalter (B4 bis B7) sind ohne Bedeutung; ist einer oder sind mehrere der vier niederwertigen Schalter (B0 bis B3) ein, so blinke die Ausgabe mit 1 Hz.

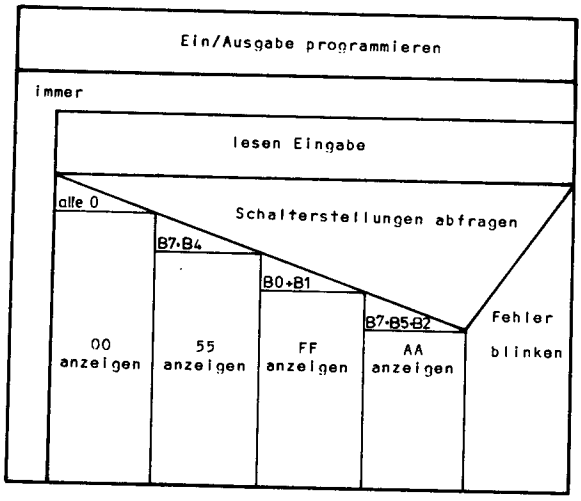


2.Aufgabe

An einer Parallelschnittstelle liegt eine Schaltung nach Bild 48 mit acht Kippschaltern und acht Anzeigeeinheiten.

a. Alle Schalter 0: Anzeige dauernd 0
b. B7 und B4 ein: Anzeige dauernd 55
c. B0 oder B1 ein: Anzeige dauernd FF
d. B7 und B5 und B2 ein: Anzeige dauernd AA

Alle nicht aufgeführten Fälle ergeben als Fehlermeldung die Anzeige FF mit 1 Hz blinkend.

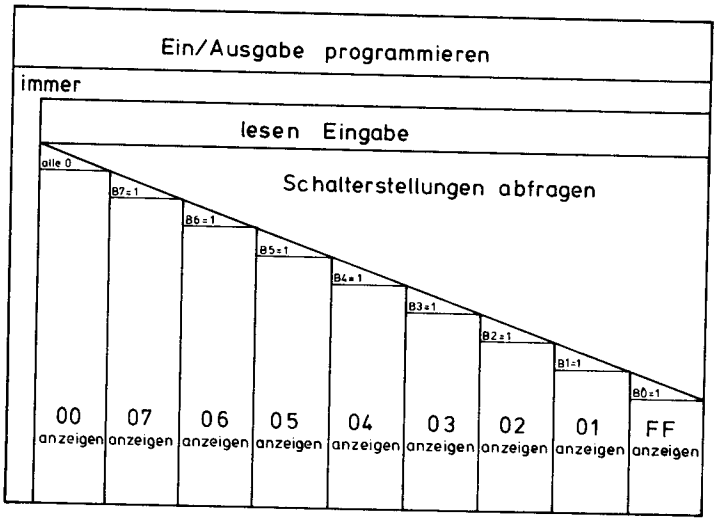


3.Aufgabe

An einer Parallelschnittstelle liegt eine Schaltung nach Bild 48 mit acht Kippschaltern und acht Anzeigeeinheiten.

a.Alle Schalter 0: Anzeige 0
b.Schalter B7 ein: Anzeige dual 7
c.Schalter B6 ein: Anzeige dual 6
d.Schalter B5 ein: Anzeige dual 5
e.Schalter B4 ein: Anzeige dual 4
f.Schalter B3 ein: Anzeige dual 3
g.Schalter B2 ein: Anzeige dual 2
h.Schalter B1 ein: Anzeige dual 1
i.Schalter B0 ein: Anzeige FF

Die höherwertigen Schalter haben Vorrang, wenn mehrere Schalter ein sind.



6. Die Verarbeitung von Daten A

Daten sind Schalterstellungen von Eingabeschaltern, Zahlen in verschiedenen Darstellungen und Zeichen, die von einer Tastatur oder einem Fernschreiber eingegeben werden können.

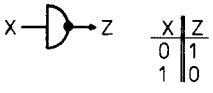
Dieser Abschnitt beschreibt verschiedene Möglichkeiten, diese Daten durch logische Befehle, Schiebebefehle oder arithmetische Befehle zu verändern. Da die eingebauten arithmetischen Befehle nur ein Byte addieren oder subtrahieren können, sind für die anderen Rechenarten und längere Zahlen mehrere Befehle erforderlich, die meist zu Unterprogrammen zusammengefaßt werden. Außerdem wird die Arbeit mit Tabellen gezeigt.

6.1 Die logischen Befehle

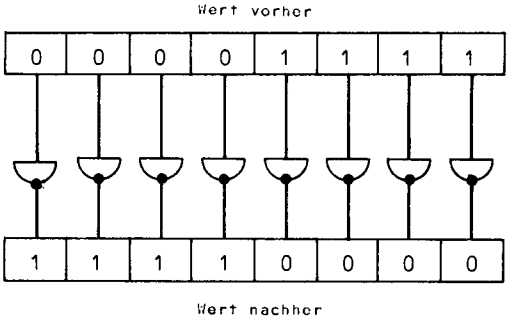
Die folgenden logischen Operationen verarbeiten ein Byte zu acht bit parallel.

a. Die NICHT-Funktion

Die Funktion bildet das 1er Komplement. Die gleiche Wirkung läßt sich auch durch das exklusive ODER mit der Konstanten 11111111 erreichen.

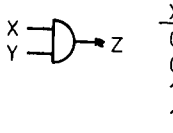


X	Z
0	1
1	0

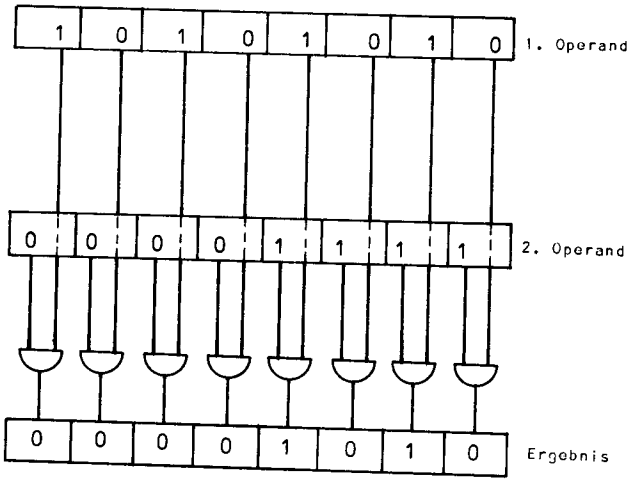


b. Die UND-Funktion

Die Funktion dient zum Ausblenden (Einbau von Nullen) bestimmter Bitpositionen mit einer Maske.



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



Beispiel:

Die vier werthöchsten Bits des Akkumulators seien ohne Rücksicht auf ihren Wert 0 zu setzen; die vier wertniedrigsten Bits sollen erhalten bleiben.

Akkumulator	10101010
Maske	00001111
Ergebnis	00001010

Das Ergebnis (logische Produkt) ist in den Bitpositionen 0, in denen die Maske 0 ist und behält dort seinen alten Wert, wo die Maske 1 ist.

c. Die ODER-Funktion

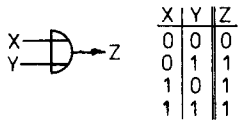
Die Funktion dient zum Einbau von 1en in bestimmte Bitpositionen mit Hilfe einer Konstanten.

Beispiel:

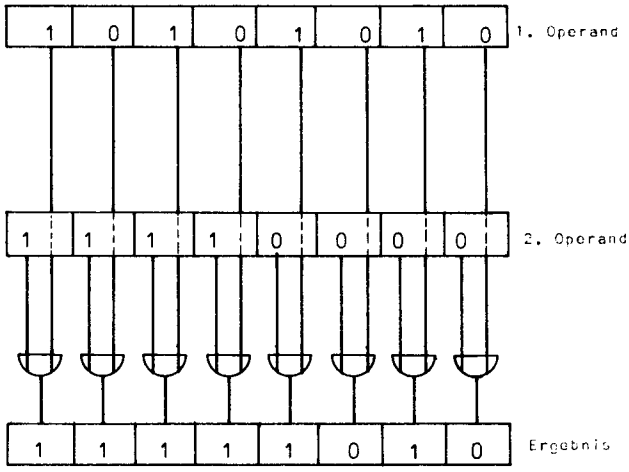
Die vier werthöchsten Bits des Akkumulators seien ohne Rücksicht auf ihren alten Wert 1 zu setzen, die vier wertniedrigsten Bits sollen erhalten bleiben.

Akkumulator	10101010
Konstante	11110000
Ergebnis	11111010

Das Ergebnis (logische Summe) ist in den Bitpositionen 1, in denen die Konstante 1 ist und behält dort seinen alten Wert, wo die Konstante 0 ist.

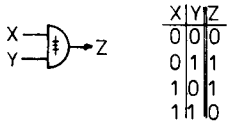


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

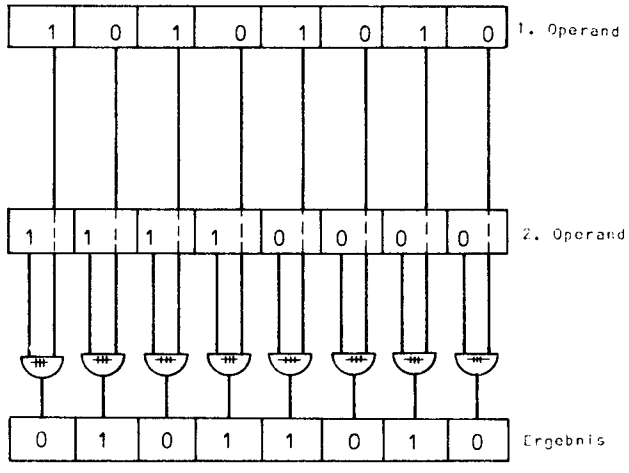


d. Die exklusive ODER-Funktion (XOR)

Die Funktion dient zum Komplementieren (aus 0 mach 1, aus 1 mach 0) bestimmter Bitpositionen mit einer Maske.



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0



Beispiel:

Die vier werthöchsten Bits des Akkumulators seien zu komplementieren, die vier wertniedrigsten sollen erhalten bleiben.

Akkumulator	10101010
Maske	11110000
Ergebnis	01011010

Das Ergebnis (logische Differenz) wird in den Bitpositionen komplementiert, in denen die Maske 1 ist und behält dort seinen alten Wert, wo die Maske 0 ist.
Mit Hilfe der XOR-Funktion ist es auch möglich festzustellen, in welchen Bitpositionen zwei Bytes nicht übereinstimmen.

Beispiel:

1. Wert	10101010	XOR
2. Wert	01101010	
Ergebnis	11000000	

Das Ergebnis der XOR-Funktion ist überall dort 1, wo beide Werte verschieden sind. Die UND-Funktion des Ergebnisses mit einem der beiden Werte stellt fest, ob der betreffende Ausgangswert eine 1 anstelle einer Null enthält.

Ergebnis des XOR	11000000	UND
1. Wert	10101010	
Ergebnis	10000000	

Der 1. Wert hat in Bitposition 7 abweichend vom 2. Wert ein 1er Bit.

Ergebnis des XOR	11000000	UND
2. Wert	01101010	
Ergebnis	01000000	

Der 2. Wert hat in Bitposition 6 abweichend vom 1. Wert ein 1er Bit. Die Bestimmung der Bitpositionen erfolgt im Programm durch Schiebebefehle und bedingte Sprünge.

6.1.1 Die logischen Befehle des KIM-1 KIM

Die folgende Tabelle zeigt die UND-Befehle mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NM	B	D	I	Z	C	
AND	# konst	unmittelbar	29	2	2	x				x		Akku ← Akku UND Konstante
AND	Adresse	absolut	2D	3	4	x				x		Akku ← Akku UND Speicher (absolut)
ANDZ	Adresse	zero-page	25	2	3	x				x		Akku ← Akku UND Speicher (zero-page)
AND	Adres,X	absol. ind.	3D	3	4/5	x				x		Akku ← Akku UND Speicher (absolut mod. X)
AND	Adres,Y	absol. ind.	39	3	4/5	x				x		Akku ← Akku UND Speicher (absolut mod. Y)
ANDZ	Adres,X	zer-pag ind.	35	2	4	x				x		Akku ← Akku UND Speicher (zero-page mod. X)
AND	(Adr,X)	indiz-indir.	21	2	6	x				x		Akku ← Akku UND Speicher (mod. X, indirekt)
AND	(Adr),Y	indir-indiz.	31	2	5/6	x				x		Akku ← Akku UND Speicher (indirekt, mod.Y)

AND bedeutet logisches UND.

Beispiel:

Der Akkumulator ist mit der Maske 00001111 zu verknüpfen.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0300	
0300	29		AND	#$0F
	0F			

Der BIT-Befehl bildet das logische UND zwischen Akkumulator und Speicherstelle und beeinflußt mit dem logischen Produkt das Z-Bit des Bedingungsregisters, ohne jedoch den Inhalt des Akkumulators zu verändern. Zusätzlich gelangen Bit 7 der Speicherstelle in das N-Bit und Bit 6 in das V-Bit des Bedingungsregisters. Damit kann ein Eingaberegister abgefragt werden, ohne daß der Inhalt in den Akkumulator gebracht werden muß.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	N	V	B	D	I	Z	C	
BIT	Adresse	zero-page	24	2	3	X	X				X		N=Bit7 , V=Bit6 , Z=1 für (Akku AND Sp.)=0
BIT	Adresse	absolut	2C	3	4	X	X				X		N=Bit7 , V=Bit6 , Z=1 für (Akku AND Sp.)=0

BIT bedeutet BIT-Test.

Ein Beispiel enthält der Abschnitt 5.1.1. über Programmverzweigungen.

Die folgende Tabelle zeigt die ODER-Befehle mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	N	V	B	D	I	Z	C	
ORA	# konst	unmittelbar	09	2	2	X					X		Akku ← Akku ODER Konstante
ORA	Adresse	absolut	0D	3	4	X					X		Akku ← Akku ODER Speicher (absolut)
ORAZ	Adresse	zero-page	05	2	3	X					X		Akku ← Akku ODER Speicher (zero-page)
ORA	Adres,X	absol. ind.	1D	3	4/5	X					X		Akku ← Akku ODER Speicher (absolut mod. X)
ORA	Adres,Y	absol. ind.	19	3	4/5	X					X		Akku ← Akku ODER Speicher (absolut mod. Y)
ORAZ	Adres,X	zer-pag ind.	15	2	4	X					X		Akku ← Akku ODER Speicher (zero-page mod X)
ORA	(Adr,X)	indiz-indir.	01	2	6	X					X		Akku ← Akku ODER Speicher (mod. X indirekt)
ORA	(Adr),Y	indir-indiz.	11	2	5/6	X					X		Akku ← Akku ODER Speicher (indirekt mod. Y)

ORA bedeutet <u>O</u>R <u>A</u>ccumulator = logisches ODER des Akkumulators.

Beispiel:

Der Akkumulator ist mit der Konstanten 11110000 durch die ODER-Funktion zu verknüpfen.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0300	
0300	09		ORA	#$F0
	F0			

Die folgende Tabelle zeigt das exklusive ODER (XOR) mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
EOR	# konst	unmittelbar	49	2	2	X				X		Akku ← Akku XOR Konstante
EOR	Adresse	absolut	4D	3	4	X				X		Akku ← Akku XOR Speicher (absolut)
EORZ	Adresse	zero-page	45	2	3	X				X		Akku ← Akku XOR Speicher (zero-page)
EOR	Adres,X	absol. ind.	5D	3	4/5	X				X		Akku ← Akku XOR Speicher (absolut mod. X)
EOR	Adres,Y	absol. ind.	59	3	4/5	X				X		Akku ← Akku XOR Speicher (absolut mod. Y)
EORZ	Adres,X	zer-pag ind.	55	2	4	X				X		Akku ← Akku XOR Speicher (zero-page mod. X)
EOR	(Adr,X)	indiz-indir.	41	2	6	X				X		Akku ← Akku XOR Speicher (mod. X indirekt)
EOR	(Adr),Y	indir-indiz.	51	2	5/6	X				X		Akku ← Akku XOR Speicher (indirekt mod. Y)

EOR bedeutet Exklusives OdeR

Da der Mikroprozessor 6502 des KIM keinen Befehl zum Komplementieren des Akkumulators hat, muß dies mit dem EOR-Befehl und der Konstanten 11111111 = FF ausgeführt werden.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0300	
0300	49		EOR	#$FF
	FF			

6.1.2 Die logischen Befehle des 6800

6800

Die folgende Tabelle zeigt die NICHT-Befehle.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N	Z	V	C	
COM A		code-eigen	43	1	2		X	X	01	Akku A ← Akku A negiert (1er Komplement)
COM B		code-eigen	53	1	2		X	X	01	Akku B ← Akku B negiert (1er Komplement)
COM	Adresse	absolut	73	3	6		X	X	01	Speicher ← Speicher negiert (extended)
COM	konst,X	indiziert	63	2	7		X	X	01	Speicher ← Speicher negiert (indexed)

COM bedeutet COMplementiere im 1er-Komplement.

Beispiel:

Der Akkumulator A ist zu komplementieren.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0180
0180	43		COM A	

Die folgende Tabelle zeigt die UND-Befehle.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N	Z	V	C	
AND A	# konst	unmittelbar	84	2	2		X	X		Akku A ← Akku A UND Konstante
AND A	Adresse	absolut	B4	3	4		X	X		Akku A ← Akku A UND Speicher (extended)
ANDZ A	Adresse	zero-page	94	2	3		X	X		Akku A ← Akku A UND Speicher (direct)
AND A	konst,X	indiziert	A4	2	5		X	X		Akku A ← Akku A UND Speicher (indexed)
AND B	# konst	unmittelbar	C4	2	2		X	X		Akku B ← Akku B UND Konstante
AND B	Adresse	absolut	F4	3	4		X	X		Akku B ← Akku B UND Speicher (extended)
ANDZ B	Adresse	zero-page	D4	2	3		X	X		Akku B ← Akku B UND Speicher (direct)
AND B	konst,X	indiziert	E4	2	5		X	X		Akku B ← Akku B UND Speicher (indexed)

AND bedeutet logisches UND .

Beispiel:

Der Akkumulator A ist mit der Maske 00001111 zu verknüpfen.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name    Befehl    Operand

                        ORG       $0180
0180     84             AND A     #$0F
         0F
```

Der BIT-Befehl bildet das logische UND zwischen einem Akkumulator und einem Vergleichswert, ohne den Inhalt des Registers oder Speichers zu verändern. Ähnlich wie bei den Vergleichsbefehlen wird nur das Bedingungsregister beeinflußt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI	NZ	VC	
BIT A	# konst	unmittelbar	85	2	2		X	X0	Akku A UND Konstante
BIT A	Adresse	absolut	B5	3	4		X	X0	Akku A UND Speicher (extended)
BITZ A	Adresse	zero-page	95	2	3		X	X0	Akku A UND Speicher (direct)
BIT A	konst,X	indiziert	A5	2	5		X	X0	Akku A UND Speicher (indexed)
BIT B	# konst	unmittelbar	C5	2	2		X	X0	Akku B UND Konstante
BIT B	Adresse	absolut	F5	3	4		X	X0	Akku B UND Speicher (extended)
BITZ B	Adresse	zero-page	D5	2	3		X	X0	Akku B UND Speicher (direct)
BIT	konst,X	indiziert	E5	2	5		X	X0	Akku B UND Speicher (indexed)

BIT bedeutet BIT-Test.

Beispiel:

Das Eingaberegister EIN ist auf Null zu prüfen. Die werthöchste Bitposition B7 wird mit der Maske 0111 1111 = $7F ausgeblendet.

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name    Befehl    Operand

                EIN     EQU       $8006

                        ORG       $0180
0180     86             LDA A     #$7F
         7F
0182     B5             BIT A     EIN
         80
         06
```

Die folgende Tabelle zeigt die ODER-Befehle mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI	NZ	VC	
ORA A	# konst	unmittelbar	8A	2	2		X	X0	Akku A ← Akku A ODER Konstante
ORA A	Adresse	absolut	BA	3	4		X	X0	Akku A ← Akku A ODER Speicher (extended)
ORAZ A	Adresse	zero-page	9A	2	3		X	X0	Akku A ← Akku A ODER Speicher (direct)
ORA A	konst,X	indiziert	AA	2	5		X	X0	Akku A ← Akku A ODER Speicher (indexed)
ORA B	# konst	unmittelbar	CA	2	2		X	X0	Akku B ← Akku B ODER Konstante
ORA B	Adresse	absolut	FA	3	4		X	X0	Akku B ← Akku B ODER Speicher (extended)
ORAZ B	Adresse	zero-page	DA	2	3		X	X0	Akku B ← Akku B ODER Speicher (direct)
ORA B	konst,X	indiziert	EA	2	5		X	X0	Akku B ← Akku B ODER Speicher (indexed)

ORA bedeutet OR Accumulator = logisches ODER des Akkumulators.

Beispiel:

Der Akkumulator A ist mit der Konstanten 11110000 durch die ODER-Funktion zu verknüpfen.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0180
0180	8A		ORA A	#$F0
	F0			

Die folgende Tabelle zeigt das exklusive ODER (XOR) mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI	NZVC	
EOR A	# konst	unmittelbar	88	2	2		xxx0	Akku A ← Akku A XOR Konstante
EOR A	Adresse	absolut	B8	3	4		xxx0	Akku A ← Akku A XOR Speicher (extended)
EORZ A	Adresse	zero-page	98	2	3		xxx0	Akku A ← Akku A XOR Speicher (direct)
EOR A	konst,X	indiziert	A8	2	5		xxx0	Akku A ← Akku A XOR Speicher (indexed)
EOR B	# konst	unmittelbar	C8	2	2		xxx0	Akku B ← Akku B XOR Konstante
EOR B	Adresse	absolut	F8	3	4		xxx0	Akku B ← Akku B XOR Speicher (extended)
EORZ B	Adresse	zero-page	D8	2	3		xxx0	Akku B ← Akku B XOR Speicher (direct)
EOR B	konst,X	indiziert	E8	2	5		xxx0	Akku B ← Akku B XOR Speicher (indexed)

EOR bedeutet Exklusives OdeR.

Beispiel:

Die vier werthöchsten Bits des Akkumulators A sind zu komplementieren, die vier wertniedrigsten bleiben erhalten; die Maske lautet 11110000.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0180
0180	88		EOR A	#$F0
	F0			

6.1.3 Die logischen Befehle des 8080/8085

8080

Der folgende Befehl komplementiert den Akkumulator ohne Veränderung des Bedingungsregisters.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
CMA		code-eigen	2F	1	4			Akku ← Akku negiert (1er Komplement)

CMA bedeutet CoMplementiere Akkumulator.

Beispiel:

Komplementieren des Akkumulators.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	2F		CMA	

Die folgende Tabelle zeigt die UND-Befehle mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ	H	P	C	
ANI	konst	unmittelbar	E6	2	7	xx	x	x	x	Akku ← Akku UND Konstante
ANA	A	code-eigen	A7	1	4	xx	x	x	x	Akku ← Akku UND Akku
ANA	B	code-eigen	A0	1	4	xx	x	x	x	Akku ← Akku UND B-Register
ANA	C	code-eigen	A1	1	4	xx	x	x	x	Akku ← Akku UND C-Register
ANA	D	code-eigen	A2	1	4	xx	x	x	x	Akku ← Akku UND D-Register
ANA	E	code-eigen	A3	1	4	xx	x	x	x	Akku ← Akku UND E-Register
ANA	H	code-eigen	A4	1	4	xx	x	x	x	Akku ← Akku UND H-Register
ANA	L	code-eigen	A5	1	4	xx	x	x	x	Akku ← Akku UND L-Register
ANA	M	Registerpaar	A6	1	7	xx	x	x	x	Akku ← Akku UND Speicher (Adresse in H,L)

ANI bedeutet ANd Immediate = logisches UND mit unmittelbar folgender Konstanten
ANA bedeutet ANd Accumulator = logisches UND des Akkumulators.

Beispiel:

Der Akkumulator ist mit der Maske 00001111 zu verknüpfen.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	E6		ANI	0FH
	0F			

Die folgende Tabelle zeigt die ODER-Befehle mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ	H	P	C	
ORI	konst	unmittelbar	F6	2	7	xx	x	x	x	Akku ← Akku ODER Konstante
ORA	A	code-eigen	B7	1	4	xx	x	x	x	Akku ← Akku ODER Akku
ORA	B	code-eigen	B0	1	4	xx	x	x	x	Akku ← Akku ODER B-Register
ORA	C	code-eigen	B1	1	4	xx	x	x	x	Akku ← Akku ODER C-Register
ORA	D	code-eigen	B2	1	4	xx	x	x	x	Akku ← Akku ODER D-Register
ORA	E	code-eigen	B3	1	4	xx	x	x	x	Akku ← Akku ODER E-Register
ORA	H	code-eigen	B4	1	4	xx	x	x	x	Akku ← Akku ODER H-Register
ORA	L	code-eigen	B5	1	4	xx	x	x	x	Akku ← Akku ODER L-Register
ORA	M	Registerpaar	B6	1	7	xx	x	x	x	Akku ← Akku ODER Speicher (Adresse in H,L)

ORA bedeutet OR Accumulator = logisches ODER des Akkumulators.

Beispiel:

Der Akkumulator ist mit der Konstanten 11110000 durch die ODER-Funktion zu verknüpfen.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	F6		ORI	0F0H
	F0			

Die folgende Tabelle zeigt die exklusiven ODER-Befehle (XOR) mit allen Adressierungsarten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ	H	P	C	
XRI	konst	unmittelbar	EE	2	7	XX	X	X	X	Akku ← Akku XOR Konstante
XRA	A	code-eigen	AF	1	4	XX	X	X	X	Akku ← Akku XOR Akku
XRA	B	code-eigen	A8	1	4	XX	X	X	X	Akku ← Akku XOR B-Register
XRA	C	code-eigen	A9	1	4	XX	X	X	X	Akku ← Akku XOR C-Register
XRA	D	code-eigen	AA	1	4	XX	X	X	X	Akku ← Akku XOR D-Register
XRA	E	code-eigen	AB	1	4	XX	X	X	X	Akku ← Akku XOR E-Register
XRA	H	code-eigen	AC	1	4	XX	X	X	X	Akku ← Akku XOR H-Register
XRA	L	code-eigen	AD	1	4	XX	X	X	X	Akku ← Akku XOR L-Register
XRA	M	Registerpaar	AE	1	7	XX	X	X	X	Akku ← Akku XOR Speicher (Adresse in H,L)

XRA bedeutet XOR Accumulator =exklusives ODER des Akkumulators.

Die Funktion ergibt an den Stellen Null, an denen die Bits der beiden Operanden übereinstimmen. Sie kann daher zum Löschen des Akkumulators mit einem 1-Byte-Befehl verwendet werden.

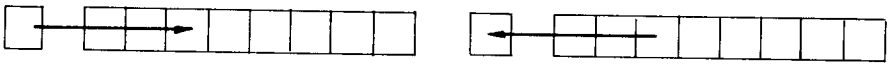
Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	AF		XRA	A

6.2 Die Schiebebefehle

A

Die Schiebebefehle verschieben den Inhalt eines Registers oder einer Speicherstelle um ein Bit zusammen mit dem Carry-Bit. Man unterscheidet:

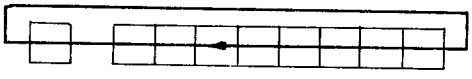
a. Die Schieberichtung



rechts schieben links schieben

b. Zyklisches Schieben (rotieren)

zyklisch rechts

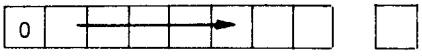


zyklisch links

c. Arithmetisches Schieben

arithmetisch rechts arithmetisch links

d. Logisches Schieben



logisch rechts

Beim arithmetischen Rechtsschieben wird Bitposition 6 gleich Bitposition 7, dem Vorzeichen in der 2er Komplement-Darstellung von Zahlen. Ist das Vorzeichen positiv, so wird ein 0er Bit nachgezogen, ist es negativ, so wird ein 1er Bit nachgezogen. Die Zahlendarstellung bleibt damit erhalten.
Die Schiebebefehle erlauben den Zugriff auf einzelne Bits eines Bytes. Der Abschnitt 5.3. zeigte die Anwendung der Schiebebefehle für Programmverzweigungen durch getrenntes Abfragen einzelner Bits und durch Aufbau einer Sprungleiste.
Weitere Anwendungen sind das Trennen und Zusammenfügen bestimmter Teile eines Bytes.

Beispiel:

Ein Register enthalte zwei BCD-codierte Dezimalziffern, die zu trennen sind.

Ausgangswert	1000 1001	Ziffern 8 und 9
UND-Maske 0F	0000 1001	rechte Ziffer 9
4 Bit log. rechts	0000 1000	linke Ziffer 8

Beispiel:

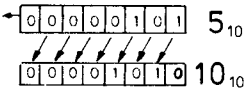
Zwei BCD-codierte Dezimalziffern in zwei Speicherstellen sollen in einem Byte zusammengefaßt werden.

Laden einer Ziffer	0000 1000	linke Ziffer 8
4 Bit ari. links	1000 0000	linke Ziffer verschoben
ODER mit 2.Ziffer	1000 1001	Ziffern 8 und 9

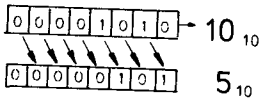
Weitere Anwendungen sind die Multiplikation mit 2 bzw. die Division durch 2 ohne Rest.

Beispiel:

Arithmetisches Verschieben um ein Bit nach links bedeutet eine Multiplikation mit 2.



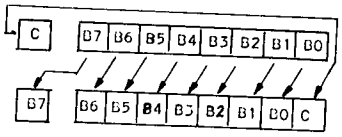
Arithmetisches Verschieben um ein Bit nach rechts bedeutet eine Division durch zwei ohne Rest.



6.2.1 Die Schiebebefehle des KIM-1

KIM

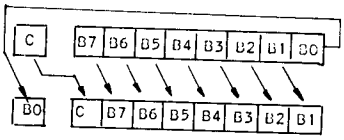
Die folgenden Befehle verschieben den Akkumulator bzw. eine Speicherstelle mit dem Carry-Bit zyklisch um ein Bit nach links.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
ROLA		code-eigen	2A	1	2	X			XX	Schiebe C und Akku 1 Bit zykl. links
ROL	Adresse	absolut	2E	3	6	X			XX	Schiebe C und Speicher (absolut)
ROLZ	Adresse	zero-page	26	2	5	X			XX	Schiebe C und Speicher (zero-page)
ROL	Adres,X	absol. ind.	3E	3	7	X			XX	Schiebe C und Speicher (absolut mod. X)
ROLZ	Adres,X	zer-pag ind.	36	2	6	X			XX	Schiebe C und Speicher (zero-page mod. X)

ROL bedeutet ROtieren Links

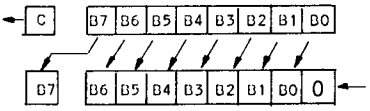
Die folgenden Befehle verschieben den Akkumulator bzw. eine Speicherstelle zusammen mit dem Carry-Bit zyklisch um ein Bit nach rechts.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
RORA		code-eigen	6A	1	2	X			XX	Schiebe C und Akku 1 Bit zykl. rechts
ROR	Adresse	absolut	6E	3	6	X			XX	Schiebe C und Speicher (absolut)
RORZ	Adresse	zero-page	66	2	5	X			XX	Schiebe C und Speicher (zero-page)
ROR	Adres,X	absol. ind.	7E	3	7	X			XX	Schiebe C und Speicher (absolut mod. X)
RORZ	Adres,X	zer-pag ind.	76	2	6	X			XX	Schiebe C und Speicher (zero-page mod. X)

ROR bedeutet ROtieren Rechts

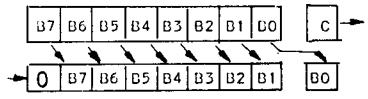
Die folgenden Befehle verschieben den Akkumulator bzw. eine Speicherstelle mit dem Carry-Bit arithmetisch um ein Bit nach links. Das Carry-Bit geht verloren, Bitposition B0 wird Null gesetzt.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
ASLA		code-eigen	0A	1	2	X			XX	Schiebe C und Akku 1 Bit arithmetisch links
ASL	Adresse	absolut	0E	3	6	X			XX	Schiebe C und Speicher (absolut)
ASLZ	Adresse	zero-page	06	2	5	X			XX	Schiebe C und Speicher (zero-page)
ASL	Adres,X	absol. ind.	1E	2	7	X			XX	Schiebe C und Speicher (absolut mod. X)
ASLZ	Adres,X	zer-pag ind.	16	2	6	X			XX	Schiebe C und Speicher (zero-page mod. X)

ASL bedeutet Arithmetisches Schieben Links

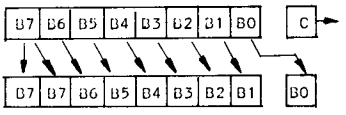
Die folgenden Befehle verschieben den Akkumulator bzw. eine Speicherstelle mit dem Carry-Bit logisch um ein Bit nach rechts. Das Carry-Bit geht verloren, Bitposition B7 wird Null gesetzt.



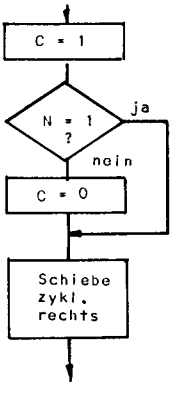
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
LSRA		code-eigen	4A	1	2	0			XX	Schiebe C und Akku 1 Bit logisch rechts
LSR	Adresse	absolut	4E	3	6	0			XX	Schiebe C und Speicher (absolut)
LSRZ	Adresse	zero-page	46	2	5	0			XX	Schiebe C und Speicher (zero-page)
LSR	Adres,X	absol. ind.	5E	2	7	0			XX	Schiebe C und Speicher (absolut mod. X)
LSRZ	Adres,X	zer-pag ind.	56	2	6	0			XX	Schiebe C und Speicher (zero-page mod. X)

LSR bedeutet Logisches Schieben Rechts

Das arithmetische Rechtsschieben, bei dem Bit 7 erhalten bleibt, ist nicht vorhanden.



Der Befehl zum arithmetischen Schieben des Akkumulators kann durch folgendes Programmstück nachgebildet werden:




```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl    Operand

                       *=$0300
0300    38             SEC
0301    30             BMI       MAX
        01
0303    18             CLC
0304    6A     MAX     RORA
```

Beispiel:

Division durch 2 mit Vorzeichen

Akkumulator vorher 11110110 = F6 = -10
Akkumulator nachher 11111011 = FB = -5

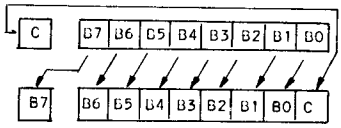
6.2.2 Die Schiebebefehle des 6800

6800

Das V-Bit (OVerflow = Überlauf) ist das exklusive ODER des N- und C-Bits nach der Verschiebung.

N	C	V
0	0	0
0	1	1
1	0	1
1	1	0

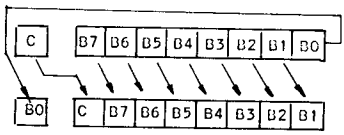
Die folgenden Befehle verschieben einen Akkumulator bzw. eine Speicherstelle zusammen mit dem Carry-Bit zyklisch um ein Bit nach links.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
ROL A		code-eigen	49	1	2	XXXX	Schiebe C und Akku A 1 Bit zykl. links
ROL B		code-eigen	59	1	2	XXXX	Schiebe C und Akku B 1 Bit zykl. links
ROL	Adresse	absolut	79	3	6	XXXX	Schiebe C und Speicher (extended)
ROL	konst,X	indiziert	69	2	7	XXXX	Schiebe C und Speicher (indexed)

ROL bedeutet ROtieren Links

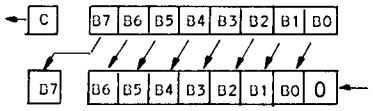
Die folgenden Befehle verschieben einen Akkumulator bzw. eine Speicherstelle zusammen mit dem Carry-Bit zyklisch um ein Bit nach rechts.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
ROR A		code-eigen	46	1	2	XXXX	Schiebe C und Akku A 1 Bit zykl. rechts
ROR B		code-eigen	56	1	2	XXXX	Schiebe C und Akku B 1 Bit zykl. rechts
ROR	Adresse	absolut	76	3	6	XXXX	Schiebe C und Speicher (extended)
ROR	konst,X	indiziert	66	2	7	XXXX	Schiebe C und Speicher (indexed)

ROR bedeutet ROtieren Rechts

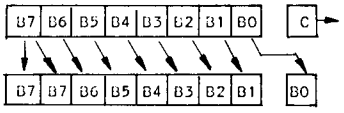
Die folgenden Befehle verschieben einen Akkumulator bzw. eine Speicherstelle zusammen mit dem Carry-Bit arithmetisch um ein Bit nach links. Bit B0 wird Null, das Carry-Bit geht verloren.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
ASL A		code-eigen	48	1	2	↕↕↕↕	Schiebe C und Akku A 1 Bit arithmetisch li.
ASL B		code-eigen	58	1	2	↕↕↕↕	Schiebe C und Akku B 1 Bit arithmetisch li.
ASL	Adresse	absolut	78	3	6	↕↕↕↕	Schiebe C und Speicher (extended)
ASL	konst,X	indiziert	68	2	7	↕↕↕↕	Schiebe C und Speicher (indexed)

ASL bedeutet Arithmetisches Schieben Links

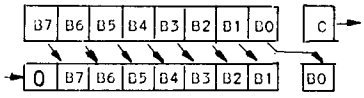
Die folgenden Befehle verschieben einen Akkumulator bzw. eine Speicherstelle zusammen mit dem Carry-Bit arithmetisch um ein Bit nach rechts. Das Vorzeichenbit B7 bleibt erhalten, das Carry-Bit geht verloren.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
ASR A		code-eigen	47	1	2	↕↕↕↕	Schiebe C und Akku A 1 Bit arithmetisch re.
ASR B		code-eigen	57	1	2	↕↕↕↕	Schiebe C und Akku B 1 Bit arithmetisch re.
ASR	Adresse	absolut	77	3	6	↕↕↕↕	Schiebe C und Speicher (extended)
ASR	konst,X	indiziert	67	2	7	↕↕↕↕	Schiebe C und Speicher (indexed)

ASR bedeutet Arithmetisches Schieben Rechts

Die folgenden Befehle verschieben einen Akkumulator bzw. eine Speicherstelle zusammen mit dem Carry-Bit logisch um ein Bit nach rechts. Das Vorzeichenbit B7 wird Null, das Carry-Bit geht verloren.



Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
LSR A		code-eigen	44	1	2	0↕↕↕	Schiebe C und Akku A 1 Bit logisch rechts
LSR B		code-eigen	54	1	2	0↕↕↕	Schiebe C und Akku B 1 Bit logisch rechts
LSR	Adresse	absolut	74	3	6	0↕↕↕	Schiebe C und Speicher (extended)
LSR	konst,X	indiziert	64	2	7	0↕↕↕	Schiebe C und Speicher (indexed)

LSR bedeutet Logisches Schieben Rechts

Beispiel:

Der Inhalt des Akkumulators A ist durch zweimaliges Schieben nach links mit 4 zu multiplizieren.

Zahlenbeispiel:

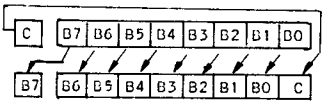
vorher 00000010 = 2
nachher 00001000 = 8

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0180
0180	48		ASL A	
0181	48		ASL A	

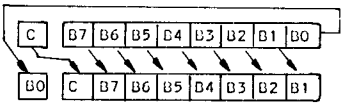
6.2.3 Die Schiebebefehle des 8080/8085 8080

Bei den Schiebebefehlen der Prozessoren 8080 und 8085 wird nur das C-Bit, nicht aber das N-Bit verändert!

a. Zyklisches Schieben des Akkumulators und Carry-Bits

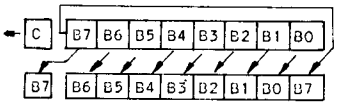


Akkumulator und Carry links

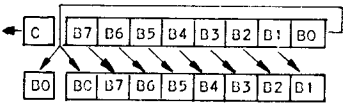


Akkumulator und Carry rechts

b. Zyklisches Schieben des Akkumulators allein



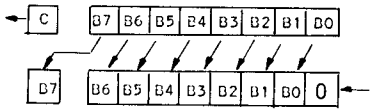
Akkumulator links C = B7



Akkumulator rechts C=B0

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N Z H P C	
RAL		code-eigen	17	1	4	X	Schiebe Akku und C 1 Bit zyklisch links
RAR		code-eigen	1F	1	4	X	Schiebe Akku und C 1 Bit zyklisch rechts
RLC		code-eigen	07	1	4	X	Schiebe Akku zyklisch links C = Bit 7
RRC		code-eigen	0F	1	4	X	Schiebe Akku zyklisch rechts C = Bit 0

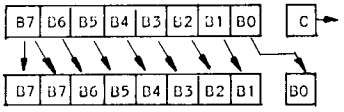
Das arithmetische Linksschieben ist nicht enthalten.



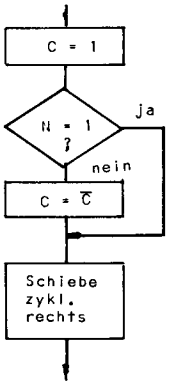
Die folgenden Befehle schieben Akkumulator und Carry-Bit zunächst zyklisch nach links und setzen Bit 0 durch eine Maske auf 0.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	17		RAL	
2881	E6		ANI	0FEH
	FE			

Das arithmetische Rechtsschieben ist nicht im Befehlssatz enthalten.

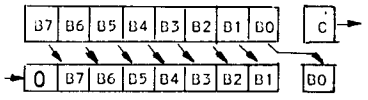


Dafür können folgende Befehle verwendet werden:



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	37		STC	
2881	FA		JM	MAX
	85			
	28			
2884	3F		CMC	
2885	1F	MAX	RAR	

Das logische Rechtsschieben ist nicht im Befehlssatz enthalten.



Die folgenden Befehle schieben den Akkumulator und das Carry-Bit zunächst zyklisch nach rechts und fügen die 0 in Bitposition B7 durch eine Maske ein.

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	1F		RAR	
2881	E6		ANI	0EFH
	EF			

6.3 Die arithmetischen Befehle A

In den acht Bits eines Registers oder einer Speicherstelle können folgende Zahlenarten dargestellt werden:

a. Positive Dualzahlen

Das werthöchste Bit des Bytes ist Bestandteil der Zahl und kein Vorzeichen. Das N-Bit des Bedingungsregisters ist bedeutungslos. Der Zahlenbereich reicht von 0 bis $+255_{10}$

Beispiel: $10001000_2 = 88_{16} = 136_{10}$

b. Dualzahlen mit Vorzeichen

Das werthöchste Bit (= N-Bit des Bedingungsregisters) gibt das Vorzeichen an, die Zahl wird im 2er Komplement dargestellt. Der Zahlenbereich reicht von -128_{10} bis $+127_{10}$

Beispiel: $10001000_2 = 88_{16} = -120_{10}$

c. BCD-codierte Dezimalzahlen

Die vier werthöchsten Bits und die vier wertniedrigsten Bits bilden jeweils eine Ziffer. Der Zahlenbereich reicht bei positiven Zahlen von 00 bis 99_{10}. Negative Zahlen müssen besonders behandelt werden.

Beispiel: $10001000_2 = 88_{10}$

Die arithmetisch-logische Einheit rechnet dual; bei Dezimalzahlen sind Korrekturen erforderlich.

Die hier behandelten Mikroprozessoren können nur addieren und subtrahieren in den genannten drei Zahlendarstellungen. Für größere Zahlen und andere Rechenarten (Multiplikation, Division und reelle Zahlen) sind mehrere Befehle erforderlich, die meist zu Unterprogrammen zusammengefaßt werden.

6.3.1 Die arithmetischen Befehle des KIM-1 KIM

Vor der Anwendung der arithmetischen Befehle muß entweder die Dezimalarithmetik (Korrekturaddierer) oder die Dualarithmetik eingeschaltet werden. Der gewählte Zustand bleibt solange erhalten, bis er durch einen der beiden folgenden Befehle wieder verändert wird.

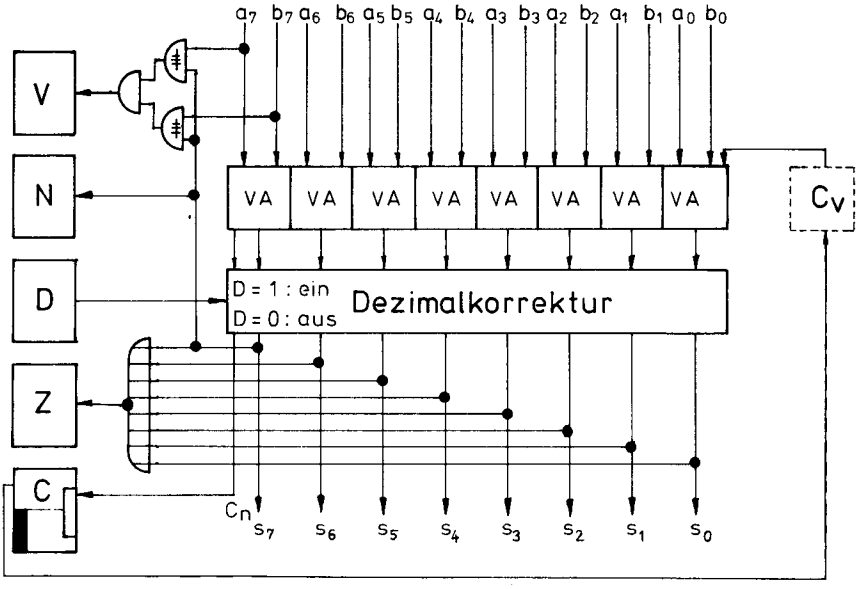
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N V B D I Z C	
CLD		code-eigen	D8	1	2	0	Dualarithmetik eingeschaltet
SED		code-eigen	F8	1	2	1	Dezimalarithmetik eingeschaltet

CLD bedeutet CLear Decimal = lösche Dezimalarithmetik.
SED bedeutet SEt Decimal = setze Dezimalarithmetik.

Das folgende Testprogramm untersucht die arithmetischen Befehle und rettet den Stand des Bedingungsregisters.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       *=$0000
0000    XX      OP1    .BYTE    $XX
0001    YY      OP2    .BYTE    $YY
0002            ERG    *=*+1
0003            BED    *=*+1

                       *=$0300
0300    D8(F8)  START  CLD              (SED)
0301    18(38)         CLC              (SEC)
0302    AD             LDA      OP1
        00
        00
0305    6D(ED)         ADC      OP2     (SBC)
        01
        00
0308    48             PHA
0309    08             PHP
030A    68             PLA
030B    8D             STA      BED
        03
        00
030E    68             PLA
030F    8D             STA      ERG
        02
        00
0312    4C      OTTO   JMP      OTTO
        12
        03
                       .END
```

Für die Addition gilt folgendes Modell:



Das Z-Bit prüft das Ergebnis auf Null. Sind alle acht Bits Null, so ist Z = 1 (ja).

Alle Additionsbefehle addieren zum Akkumulator einen Operanden und zusätzlich das Carry-Bit. Beim Arbeiten mit Zahlen von einem byte Länge ist es zweckmäßig, vor der Addition das Carry-Bit durch den Befehl CLC zu löschen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC		
CLC		code-eigen	18	1	2					0	Carry-Bit gelöscht
ADC	# konst	unmittelbar	69	2	2	xx				xx	Akku ← Akku + C + Konstante
ADC	Adresse	absolut	6D	3	4	xx				xx	Akku ← Akku + C + Speicher (absolut)
ADCZ	Adresse	zero-page	65	2	3	xx				xx	Akku ← Akku + C + Speicher (zero-page)
ADC	Adres,X	absol. ind.	7D	3	4/5	xx				xx	Akku ← Akku + C + Speicher (absolut mod. X)
ADC	Adres,Y	absol. ind.	79	3	4/5	xx				xx	Akku ← Akku + C + Speicher (absolut mod. Y)
ADCZ	Adres,X	zer-pag ind.	75	2	4	xx				xx	Akku ← Akku + C + Speicher (zero-p. mod. X)
ADC	(Adr,X)	indiz-indir.	61	2	6	xx				xx	Akku ← Akku + C + Speicher (mod X indirekt)
ADC	(Adr),Y	indir-indiz.	71	2	5/6	xx				xx	Akku ← Akku + C + Speicher (indirekt mod Y)

CLC bedeutet CLear Carry = lösche Carry-Bit.
ADC bedeutet ADdiere mit Carry.

a.) Addieren positiver Dualzahlen

Der Zahlenbereich erstreckt sich von 0 bis +255.
Der alte Inhalt des C-Bits (Cv) wird immer auf den Eingang des wertniedrigsten Volladdierers gegeben.
Der Carry-Ausgang des werthöchsten Volladdierers ergibt den neuen Inhalt des Carry-Bits (Cn).

Cn = 0 : Das Ergebnis ist kleiner als 256, kein Zahlenüberlauf.
Cn = 1 : Das Ergebnis ist größer als 255, Zahlenüberlauf.

Das Testprogramm lieferte folgende Ergebnisse:

OP1	+ OP2	+ C =	ERG	BED	NV B DIZC	Bemerkung
01	01	0	02	34	0011 0100	1+1+0=2
FF	01	0	00	37	0011 0111	255+1+0=256 Überlauf
01	01	1	03	34	0011 0100	1+1+1=3
FF	01	1	01	35	0011 0101	255+1+1=257 Überlauf
FE	01	1	00	37	0011 0111	254+1+1=256 Überlauf

b.) Addieren vorzeichenbehafteter Dualzahlen

Der Zahlenbereich erstreckt sich von -128 bis +127.
Das Carry-Bit hat hier keine Bedeutung, wird aber immer mitaddiert. Es ist deshalb vor jeder Addition zu löschen. Das V-Bit zeigt an, ob das Ergebnis im zulässigen Bereich liegt.

V = 0 : Ergebnis im zulässigen Bereich.
V = 1 : Überlauf (Zahl größer +127) oder Unterlauf (Zahl kleiner -128).

Das N-Bit zeigt das Vorzeichen des Ergebnisses an.
N = 0 : Ergebnis positiv
N = 1 : Ergebnis negativ

Das Testprogramm lieferte folgende Ergebnisse:

OP1	+	OP2	+	C	=	ERG	BED	NV B DIZC	Bemerkung
01		FF		0		00	37	0011 0111	(+1)+(-1)=0
FF		01		0		00	37	0011 0111	(-1)+(+1)=0
02		FD		0		FF	B4	1011 0100	(+2)+(-3)=-1
FD		02		0		FF	B4	1011 0100	(-3)+(+2)=-1
7F		01		0		80	F4	1111 0100	(+127)+(+1)=+128 Überlauf
80		FF		0		7F	75	0111 0101	(-128)+(-1)=-129 Unterlauf
7F		81		0		00	37	0011 0111	(+127)+(-127)=0
81		7F		0		00	37	0011 0111	(-127)+(+127)=0

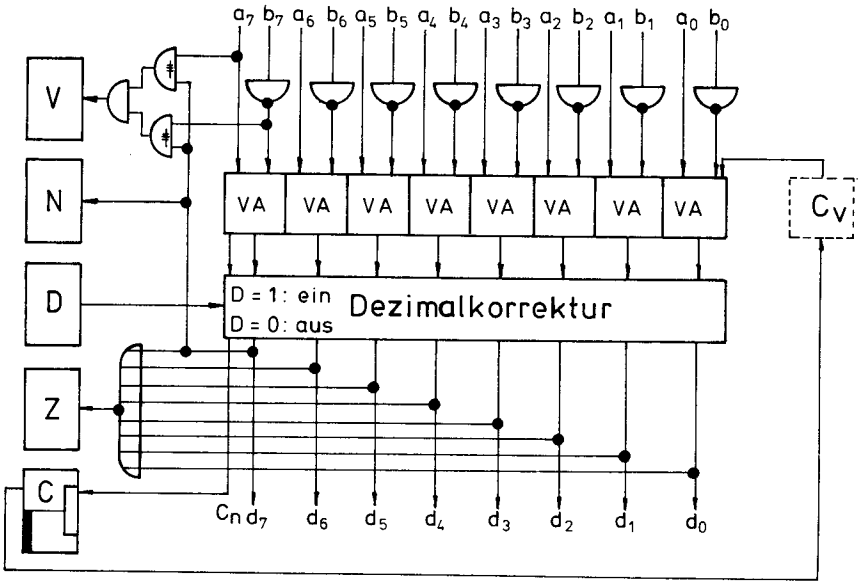
c.) Addieren von positiven BCD-Zahlen

Der Zahlenbereich erstreckt sich von 00 bis 99. Auch hier wird immer der alte Inhalt des Carry-Bits mitaddiert. Ist das neue Carry Cn = 1, so ist das Ergebnis größer als 99. Addiert man mehrstellige Zahlen, so muß der Übertrag auf die nächste Stelle addiert werden (Dekadenübertrag).

Das Testprogramm lieferte folgende Ergebnisse:

OP1	+	OP2	+	C	=	ERG	BED	NV B DIZC	Bemerkung
02		01		0		03	3C	0011 1100	2+1+0=3
09		09		0		18	3C	0011 1100	9+9+0=18
99		01		0		00	BD	1011 1101	99+1+0=100 Überlauf
99		99		0		98	7D	0111 1101	99+99+0=198 Überlauf
10		11		0		21	3C	0011 1100	10+11+0=21
01		01		1		03	3C	0011 1100	1+1+1=3
09		10		1		20	3C	0011 1100	9+10+1=20

Für die Subtraktion gilt folgendes Modell:



Die Subtraktion wird auf eine Addition des 2er-Komplementes zurückgeführt. Die Negationsschaltungen am Eingang der acht Volladdierer bilden das 1er-Komplement, zusätzlich muß über den Cv-Eingang des letzten Volladdierers eine 1 addiert werden. Der Mikroprozessor 6502 des KIM-1 berücksichtigt jedoch nicht, daß sich bei der Subtraktion die Bedeutung des Carry-Bits umdreht. Er schaltet den Inhalt des Carry-Bits direkt auf den Carry-Eingang des wertniedrigsten Volladdierers.
Für Cv = 1 wird richtig das 2er-Komplement gebildet, also zusätzlich 0 abgezogen; für Cv = 0 wird nur das 1er-Komplement gebildet, also zusätzlich 1 abgezogen.

Alle Subtraktionsbefehle subtrahieren daher vom Akkumulator einen Operanden und zusätzlich das negierte Carry-Bit. Beim Arbeiten mit Zahlen von einem byte Länge ist es zweckmäßig, vor der Subtraktion das Carry-Bit 1 zu setzen, damit $\overline{1}$ = 0 subtrahiert wird.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	D	I	Z	C	
SEC		code-eigen	38	1	2						1	Carry-Bit 1 gesetzt
SBC	# konst	unmittelbar	E9	2	2	XX					XX	Akku ← Akku - $\overline{C}$ - Konstante
SBC	Adresse	absolut	ED	3	4	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (absolut)
SBCZ	Adresse	zero-page	E5	2	3	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (zero-page)
SBC	Adres,X	absol. ind.	FD	3	4/5	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (absolut mod. X)
SBC	Adres,Y	absol. ind.	F9	3	4/5	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (absolut mod. Y)
SBCZ	Adres,X	zer-pag ind.	F5	2	4	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (zero-p. mod. X)
SBC	(Adr,X)	indiz-indir.	E1	2	6	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (mod X indirekt)
SBC	(Adr),Y	indir-indiz.	F1	2	5/6	XX					XX	Akku ← Akku - $\overline{C}$ - Speicher (indirekt mod Y)

SBC bedeutet SuBtrahiere mit Carry

a.) Subtrahieren positiver Dualzahlen

Der Zahlenbereich erstreckt sich von 0 bis +255. Ergibt sich ein negativer Wert, so spricht man von einem Zahlenunterlauf. Der Carry-Ausgang des werthöchsten Volladdierers ergibt den neuen Inhalt des Carry-Bits Cn. Wegen der Addition des Komplementes hat das Carry-Bit bei der Subtraktion folgende Bedeutung:

Cn = 1 : Das Ergebnis ist größer oder gleich 0, kein Zahlenunterlauf.
Cn = 0 : Das Ergebnis ist kleiner 0, Zahlenunterlauf.

Das Testprogramm lieferte folgende Ergebnisse:

OP1	- OP2	- $\bar{C}$	= ERG	BED	NV B DIZC	Bemerkung
02	01	C=1	01	35	0011 0101	2-1-0=1
01	01	C=1	00	37	0011 0111	1-1-0=0
01	02	C=1	FF	B4	1011 0100	1-2-0=-1 Unterlauf
03	01	C=0	01	35	0011 0101	3-1-1=1
02	01	C=0	00	37	0011 0111	2-1-1=0
01	01	C=0	FF	B4	1011 0100	1-1-0=-1 Unterlauf

b.) Subtrahieren vorzeichenbehafteter Dualzahlen

Der Zahlenbereich erstreckt sich von -128 bis +127.
Das Carry-Bit Cn hat als Ergebnis hier keine Bedeutung, das negierte Carry-Bit Cv wird zusätzlich subtrahiert. Es ist deshalb vor jeder Subtraktion auf 1 zu setzen. Das V-Bit zeigt an, ob das Ergebnis im zulässigen Bereich liegt.

V = 0 : Ergebnis im zulässigen Bereich
V = 1 : Überlauf (Zahl größer +127) oder Unterlauf (Zahl kleiner -128)

Das N-Bit zeigt das Vorzeichen des Ergebnisses an.

N = 0 : Ergebnis positiv
N = 1 : Ergebnis negativ

Das Testprogramm lieferte folgende Ergebnisse:

OP1	OP2	- $\bar{C}$ =	ERG	BED	NV B DIZC	Bemerkung
01	02	C=1	FF	B4	1011 0100	(+1)-(+2)=-1
FE	FF	C=1	FF	B4	1011 0100	(-2)-(-1)=-1
FF	FE	C=1	01	35	0011 0101	(-1)-(-2)=+1
FF	FF	C=1	00	37	0011 0111	(-1)-(-1)=0
7F	FF	C=1	80	F4	1111 0100	(+127)-(-1)=+128 Überlauf
80	01	C=1	7F	75	0111 0101	(-128)-(+1)=-129 Unterlauf
80	FF	C=1	81	B4	1011 0100	(-128)-(-1)=-127
02	FF	C=1	03	34	0011 0100	(+2)-(-1)=+3
00	00	C=1	00	37	0011 0111	(+0)-(0)=0

c.) Subtrahieren positiver BCD-Zahlen

Der Zahlenbereich erstreckt sich von 00 bis 99. Auch hier wird zusätzlich das negierte Carry-Bit subtrahiert. Ist das neue Carry Cn = 0, so ist das Ergebnis kleiner als 0. Subtrahiert man mehrstellige Zahlen, so muß von der nächsten Stelle geborgt werden.

OP1	OP2	- $\bar{C}$ =	ERG	BED	NV B DIZC	Bemerkung
02	01	1	01	3D	0011 1101	2-1-0=1
99	55	1	44	7D	0111 1101	99-55-0=44
01	02	1	99	BC	1011 1100	1-2-0=99 Unterlauf
55	99	1	56	FC	1111 1100	55-99-0=56 Unterlauf
02	01	0	00	3F	0011 1111	2-1-1=0
01	02	0	98	BC	1011 1100	1-2-1=98

6.3.2 Die arithmetischen Befehle des 6800 6800

Alle arithmetischen Befehle arbeiten in der Dualarithmetik; für BCD-codierte Dezimalzahlen ist eine nachträgliche Korrektur mit dem Befehl DAA erforderlich. Das H-Bit (Halb-Übertrag) des Bedingungsregisters wird 1 gesetzt, wenn ein Übertrag von Bit 3 nach Bit 4 auftritt. Dies ist nur für die Addition von BCD-Zahlen von Bedeutung.
Bei den arithmetischen Befehlen, die das Carry-Bit berücksichtigen, kann dieses vorher mit einem der folgenden Befehle gesetzt oder gelöscht werden:

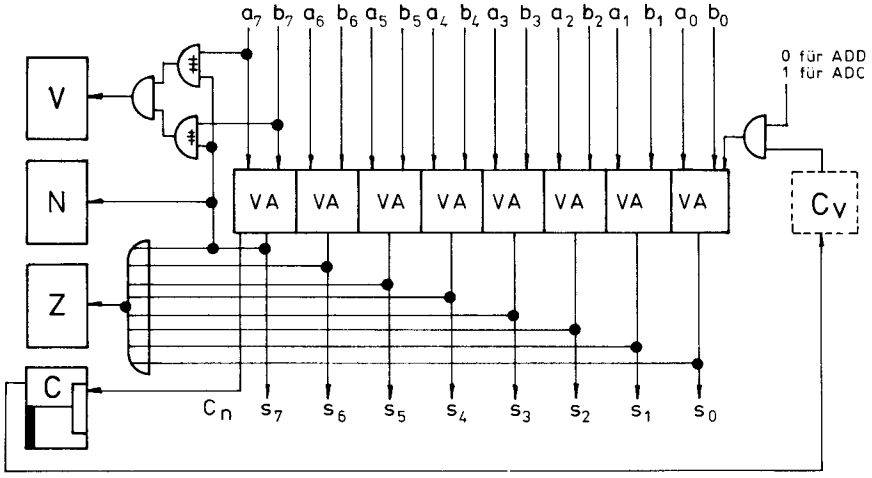
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
CLC		code-eigen	0C	1	2	0	Carry-Bit gelöscht
SEC		code-eigen	0D	1	2	1	Carry-Bit gesetzt

CLC bedeutet CLear Carry = lösche Carry-Bit
SEC bedeutet SEt Carry = setze Carry-Bit

Das folgende Testprogramm untersucht die arithmetischen Befehle:

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000    XX      ADAM   FCB      $XX
0001    YY      EVA    FCB      $YY
0002            ERG    RMB      1
0003            BED    RMB      1

                       ORG      $0100
0100    0C(0D)  START  CLC              (SEC)
0101    F6             LDA B    ADAM
        00
        00
0104    F9(F2)         ADC B    EVA     (SBC)
        00
        01
0107    01             NOP
0108    07             TPA
0109    B7             STA A    BED
        00
        03
010C    F7             STA B    ERG
        00
        02
010F    20      MAX    BRA      MAX
        FE
                       END
```

Für die Additionsbefehle gilt folgendes Modell:



Für den Befehl ADD (Addiere ohne Berücksichtigung des Carry-Bits) ist das Carry-Bit vom Addierer getrennt, der Carry-Eingang des wertniedrigsten Volladdierers ist konstant 0.

$$S = A + B$$

Das Z-Bit prüft das Ergebnis auf Null. Sind alle acht Bits Null, so ist Z = 1 (ja). Ist nur ein Bit ungleich Null, so ist Z = 0 (nein).

Die folgenden Befehle addieren zum Inhalt eines Akkumulators einen zweiten Operanden ohne Berücksichtigung des Carry-Bits.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
ABA		code-eigen	1B	1	2	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + Akku B
ADD A	# konst	unmittelbar	8B	2	2	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + Konstante
ADD A	Adresse	absolut	BB	3	4	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + Speicher (extended)
ADDZ A	Adresse	zero-page	9B	2	3	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + Speicher (direct)
ADD A	konst,X	indiziert	AB	2	5	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + Speicher (indexed)
ADD B	# konst	unmittelbar	CB	2	2	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + Konstante
ADD B	Adresse	absolut	FB	3	4	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + Speicher (extended)
ADDZ B	Adresse	zero-page	DB	2	3	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + Speicher (direct)
ADD B	konst,X	indiziert	EB	2	5	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + Speicher (indexed)

Für den Befehl ADC (Addiere mit Carry-Bit) wird der Inhalt des Carry-Bits auf den Eingang des wertniedrigsten Volladdierers gegeben und bei der Addition mit berücksichtigt.

$$S = A + B + C$$

Die folgenden Befehle addieren zum Inhalt eines Akkumulators einen zweiten Operanden mit Berücksichtigung des Carry-Bits. Sie dienen dazu, Zahlen von mehr als einem byte Länge zu addieren.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen H I N Z V C	
ADC A	# konst	unmittelbar	89	2	2	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + C + Konstante
ADC A	Adresse	absolut	B9	3	4	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + C + Speicher (extended)
ADCZ A	Adresse	zero-page	99	2	3	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + C + Speicher (direct)
ADC A	konst,X	indiziert	A9	2	5	↕ X ↕ ↕ ↕ ↕	Akku A ← Akku A + C + Speicher (indexed)
ADC B	# konst	unmittelbar	C9	2	2	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + C + Konstante
ADC B	Adresse	absolut	F9	3	4	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + C + Speicher (extended)
ADCZ B	Adresse	zero-page	D9	2	3	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + C + Speicher (direct)
ADC B	konst,X	indiziert	E9	2	5	↕ X ↕ ↕ ↕ ↕	Akku B ← Akku B + C + Speicher (indexed)

a.) Addieren positiver Dualzahlen

Der Zahlenbereich erstreckt sich von 0 bis +255.
Der Carry-Ausgang des werthöchsten Volladdierers ergibt den neuen Inhalt des Carry-Bits (Cn).

Cn = 0 : Das Ergebnis ist kleiner als 256, kein Zahlenüberlauf.
Cn = 1 : Das Ergebnis ist größer als 255, Zahlenüberlauf.

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	EVA	C	ERG	BED	--HI NZVC	Bemerkung
01	01	0	02	D0	1100 0000	1+1=2
FF	01	0	00	F5	1111 0101	255+1=256 Überlauf
01	01	1	03	D0	1100 0000	1+1+1=3
FF	01	1	01	F1	1111 0001	255+1+1=257 Überlauf
FE	01	1	00	F5	1111 0101	254+1+1=256 Überlauf

b.) Addieren vorzeichenbehafteter Dualzahlen

Der Zahlenbereich erstreckt sich von -128 bis +127.
Das Carry-Bit hat hier keine Bedeutung, wird aber beim Befehl ADC (Addiere mit Carry) mit berücksichtigt. Das V-Bit zeigt an, ob das Ergebnis im zulässigen Bereich liegt.

V = 0 : Ergebnis im zulässigen Bereich
V = 1 : Überlauf (Zahl größer +127) oder Unterlauf (Zahl kleiner -128)

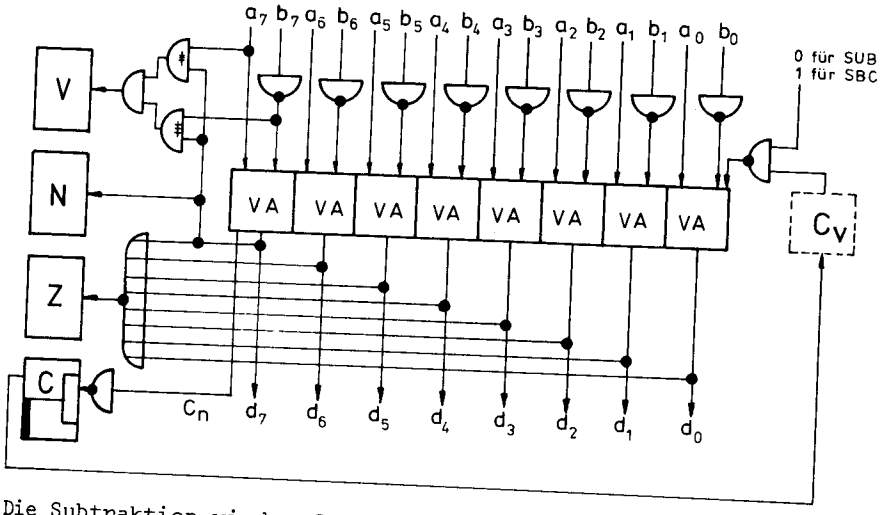
Das N-Bit gibt das Vorzeichen des Ergebnisses an.

N = 0 : Ergebnis positiv
N = 1 : Ergebnis negativ

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	EVA	C	ERG	BED	--HI NZVC	Bemerkung
01	FF	0	00	F5	1111 0101	1+(-1)= 0
FF	01	0	00	F5	1111 0101	-1+1=0
02	FD	0	FF	D8	1100 1000	2+(-3)=-1
FD	02	0	FF	D8	1100 1000	-3+2=-1
7F	01	0	80	FA	1111 1010	127+1=128 Überlauf
80	FF	0	7F	D3	1100 0011	-128+(-1)=-129 Unterlauf
7F	81	0	00	F5	1111 0101	127+(-127)=0
81	7F	0	00	F5	1111 0101	-127+127=0

Für die Subtraktion gilt folgendes Modell:



Die Subtraktion wird auf eine Addition des 2er-Komplementes zurückgeführt. Die Negationsschaltungen am Eingang der acht Volladdierer bilden das 1er-Komplement. Zusätzlich muß über den Cv-Eingang des wertniedrigsten Volladdierers eine 1 addiert werden. Da sich bei jeder Subtraktion die Bedeutung des Carry-Bits umdreht, negiert der Mikroprozessor 6800 automatisch bei jeder Subtraktion das Carry-Bit. Es wird also das negierte Carry-Bit auf den Eingang des wertniedrigsten Volladdierers gegeben und der Ausgang des werthöchsten Volladdierers negiert im Carry-Bit gespeichert.

Für den Befehl SUB (Subtrahiere ohne Carry) wird das Carry-Bit vom Addierer getrennt; der Carry-Eingang des wertniedrigsten Volladdierers ist konstant 1.

$$D = A - B$$

Die folgenden Befehle subtrahieren vom Inhalt eines Akkumulators einen zweiten Operanden ohne Berücksichtigung des Carry-Bits.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
SBA		code-eigen	10	1	2	↕↕↕↕	Akku A ← Akku A - Akku B
SUB A	# konst	unmittelbar	80	2	2	↕↕↕↕	Akku A ← Akku A - Konstante
SUB A	Adresse	absolut	B0	3	4	↕↕↕↕	Akku A ← Akku A - Speicher (extended)
SUBZ A	Adresse	zero-page	90	2	3	↕↕↕↕	Akku A ← Akku A - Speicher (direct)
SUB A	konst,X	indiziert	A0	2	5	↕↕↕↕	Akku A ← Akku A - Speicher (indexed)
SUB B	# konst	unmittelbar	C0	2	2	↕↕↕↕	Akku B ← Akku B - Konstante
SUB B	Adresse	absolut	F0	3	4	↕↕↕↕	Akku B ← Akku B - Speicher (extended)
SUBZ B	Adresse	zero-page	D0	2	3	↕↕↕↕	Akku B ← Akku B - Speicher (direct)
SUB B	konst,X	indiziert	E0	2	5	↕↕↕↕	Akku B ← Akku B - Speicher (indexed)

Für den Befehl SBC (Subtrahiere mit Carry) wird das negierte Carry-Bit = Borrow (Borgen) auf den Carry-Eingang des wertniedrigsten Volladdierers gegeben. Der SBC-Befehl dient zur grup-

penweisen Subtraktion von Dualzahlen, die mehr als acht bit lang sind. Ist bei einer Gruppe der Subtrahend (B) größer als der Minuend (A), so muß bei der folgenden Subtraktion zusätzlich eine 1 abgezogen werden (Borrow = borgen); die Addition der 1 zur Komplementbildung unterbleibt.

$$D = A - B - 1$$

Die folgenden Befehle subtrahieren vom Inhalt eines Akkumulators einen zweiten Operanden mit Berücksichtigung des Carry-Bits.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
SBC A	# konst	unmittelbar	82	2	2	xxxx	Akku A ← Akku A - C - Konstante
SBC A	Adresse	absolut	B2	3	4	xxxx	Akku A ← Akku A - C - Speicher (extended)
SBCZ A	Adresse	zero-page	92	2	3	xxxx	Akku A ← Akku A - C - Speicher (direct)
SBC A	konst,X	indiziert	A2	2	5	xxxx	Akku A ← Akku A - C - Speicher (indexed)
SBC B	# konst	unmittelbar	C2	2	2	xxxx	Akku B ← Akku B - C - Konstante
SBC B	Adresse	absolut	F2	3	4	xxxx	Akku B ← Akku B - C - Speicher (extended)
SBCZ B	Adresse	zero-page	D2	2	3	xxxx	Akku B ← Akku B - C - Speicher (direct)
SBC	konst,X	indiziert	E2	2	5	xxxx	Akku B ← Akku B - C - Speicher (indexed)

Die folgenden Befehle bilden das 2er-Komplement eines Registers oder einer Speicherstelle; sie subtrahieren von der Konstanten 00 den entsprechenden Inhalt des Registers oder der Speicherstelle.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
NEG A		code-eigen	40	1	2	xxxx	Akku A ← 00 - Akku A (2er Komplement)
NEG B		code-eigen	50	1	2	xxxx	Akku B ← 00 - Akku B (2er Komplement)
NEG	Adresse	absolut	70	3	6	xxxx	Speicher ← 00 - Speicher (extended)
NEG	konst,X	indiziert	60	2	7	xxxx	Speicher ← 00 - Speicher (indexed)

NEG heißt NEGiere.

a.) Subtrahieren positiver Dualzahlen

Der Zahlenbereich erstreckt sich von 0 bis +255. Ergibt sich ein negativer Wert, so spricht man von einem Zahlenunterlauf. Der Carry-Eingang des wertniedrigsten Volladdierers liegt beim SUB-Befehl (Subtraktion ohne Carry) konstant auf 1. Damit wird das 2er-Komplement gebildet. Beim SBC-Befehl (Subtraktion mit Carry) wird der Vorübertrag Cv automatisch negiert, so daß zusätzlich das Carry-Bit abgezogen wird.

Cv = 0 : D = A - B - 0
Cv = 1 : D = A - B - 1

Auch am Ausgang des werthöchsten Volladdierers wird bei jeder Subtraktion das Carry-Bit negiert, damit es die gleiche Bedeutung wie bei der Addition erhält.

Cn = 0 : Das Ergebnis ist größer oder gleich 0, kein Zahlenüberlauf
Cn = 1 : Das Ergebnis ist kleiner 0, Zahlenunterlauf.

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	EVA	C	= ERG	BED	--HI NZVC	Bemerkung
02	01	0	01	F0	1111 0000	2-1-0=1
01	01	0	00	F4	1111 0100	1-1-0=0
01	02	0	FF	F9	1111 1001	1-2-0=-1 Unterlauf
03	01	1	01	F0	1111 0000	3-1-1=1
02	01	1	00	F4	1111 0100	2-1-1=0
01	01	1	FF	F9	1111 1001	1-1-1=-1 Unterlauf

b.) Subtrahieren vorzeichenbehafteter Dualzahlen

Der Zahlenbereich erstreckt sich von -128 bis +127. Das Carry-Bit hat als Ergebnis keine Bedeutung, beim SBC-Befehl (Subtrahiere mit Carry) wird der Vorübertrag Cv zusätzlich subtrahiert.

Cv = 0 : D = A - B - 0
Cv = 1 : D = A - B - 1

Das V-Bit zeigt an, ob das Ergebnis im zulässigen Bereich liegt.

V = 0 : Ergebnis im zulässigen Bereich
V = 1 : Überlauf (Ergebnis größer +127) oder Unterlauf
(Ergebnis kleiner -128)

Das N-Bit zeigt das Vorzeichen des Ergebnisses an.

N = 0 : Ergebnis positiv
N = 1 : Ergebnis negativ

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	EVA	C	= ERG	BED	--HI NZVC	Bemerkung
01	02	0	FF	F9	1111 1001	1-(+2)=-1
FE	FF	0	FF	F9	1111 1001	-2-(-1)=-1
FF	FE	0	01	F0	1111 0000	-1-(-2)=+1
FF	FF	0	00	F4	1111 0100	-1-(-1)=0
7F	FF	0	80	FB	1111 1011	127-(-1)=128 Überlauf
80	01	0	7F	F2	1111 0010	-128-(+1)=-129Unterlauf
80	FF	0	81	F9	1111 1001	-128-(-1)=-127
02	FF	0	03	F1	1111 0001	+2-(-1)=+3
00	00	0	00	F4	1111 0100	0-0=0

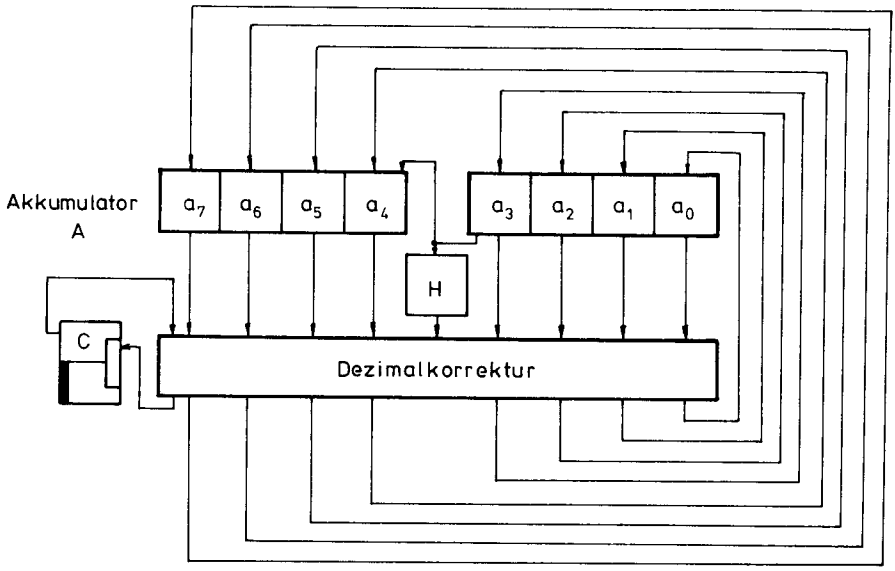
Beim Rechnen mit BCD-codierten Dezimalzahlen muß nach jedem Additionsbefehl die Dezimalkorrektur vorgenommen werden.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --H NZVC	
DAA		code-eigen	19	1	2	xx x	Dezimalkorrektur im Akku A für BCD-Zahlen

DAA = Decimal Adjust accumulator A = Dezimalkorrektur für den Akkumulator A.

Der Befehl arbeitet nur dann richtig, wenn unmittelbar vorher einer der Befehle ABA oder ADD A oder ADC A angewendet wurde und wenn die Operanden aus gültigen BCD-Ziffern im Bereich von 0000 bis 1001 bestanden.

Für den Befehl DAA (Dezimalkorrektur Akkumulator A) gilt folgendes Modell:



Bei der Arbeit mit BCD-codierten Dezimalzahlen (8-4-2-1-Code) enthält ein Byte zwei positive Dezimalziffern. In den Bitpositionen a0 bis a3 (rechtes Halbbyte) steht die wertniedrigere Ziffer, in den Bitpositionen a4 bis a7 (linkes Halbbyte) die werthöhere. Das H-Bit (Halb-Übertrag) gibt an, ob ein Übertrag von Bitposition 3 nach Bitposition 4 erfolgt, also von der wertniedrigeren Ziffer zur werthöheren. Da das H-Bit nur durch die Additionsbefehle ABA, ADD und ADC beeinflußt wird, können BCD codierte Dezimalzahlen nur addiert werden.
Das H-Bit wird automatisch durch den Korrekturbefehl DAA verwendet und kann vom Programmierer nur über eine maskierte Bearbeitung des Bedingungsregisters ausgewertet werden.

Der Zahlenbereich positiver Dezimalzahlen erstreckt sich von 00 bis +99. Auch hier wird unter Berücksichtigung des Vorübertrags Cv addiert. Ist der Ausgang des werthöchsten Volladdierers

Cn = 1, so ist das Ergebnis größer als 99. Verarbeitet man mehrstellige Zahlen, so muß der Übertrag auf die nächste Stelle addiert werden (Dekadenübertrag). Das folgende Testprogramm addiert zwei zweistellige BCD-Zahlen.

```
  Hexadezimal                Symbolisch
Adresse Inhalt   Name        Befehl    Operand

                             ORG       $0000
0000     XX      ADAM        FCB       $XX
0001     YY      EVA         FCB       $YY
0002             ERG         RMB       1
0003             BED         RMB       1

                             ORG       $0180
0180     0C(0D)  START       CLC                 (SEC)
0181     B6                  LDA A     ADAM
         00
         00
0184     B9                  ADC A     EVA
         00
         01
0187     19                  DAA
0188     36                  PSH A
0189     33                  PUL B
018A     07                  TPA
018B     B7                  STA A     BED
         00
         03
018E     F7                  STA B     ERG
         00
         02
0191     20      OTTO        BRA       OTTO
         FE
                             END
```

Es lieferte folgende Ergebnisse für den Fall Cv = 0:

ADAM +	EVA =	ERG	BED	--HI NZVC	Bemerkung
02	01	03	D0	1100 0000	2+1=3
09	09	18	F0	1111 0000	9+9=18
99	01	00	D5	1100 0101	99+1=100 Überlauf
99	99	98	FB	1111 1011	99+99=198 Überlauf
01	02	03	D0	1100 0000	1+2=3
10	11	21	D0	1100 0000	10+11=21
AA	01	11	D1	1100 0001	keine BCD-Ziffer

Da die Subtraktionsbefehle das H-Bit nicht beeinflussen, muß bei der Subtraktion positiver BCD-Zahlen im Programm das 10er-Komplement gebildet und addiert werden. Dazu ist zunächst die Differenz zur Konstanten 9 zu bilden (9er-Komplement) und bei der nachfolgenden Addition zusätzlich eine 1 zu addieren (Cv=1). Durch Vernachlässigung des Übertrags wird die Komplementbildung rückgängig gemacht.

Zahlenbeispiel:

8 - 3 => 8 + (9-3)+1 => 8 + 6 + 1 => 15 - 10 => 5

Das folgende Testprogramm untersucht dieses Verfahren für zweistellige BCD-Zahlen.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000    XX     ADAM    FCB      $XX
0001    YY     EVA     FCB      $YY
0002           ERG     RMB      1
0003           BED     RMB      1

                       ORG      $0180
0180    86     START   LDA A    #$99
        99
0182    B0             SUB A    EVA
        00
        01
0185    0D             SEC
0186    B9             ADC A    ADAM
        00
        00
0189    19             DAA
018A    36             PSH A
018B    33             PUL B
018C    07             TPA
018D    B7             STA A    BED
        00
        03
0190    F7             STA B    ERG
        00
        02
0193    20     OTTO    BRA      OTTO
        FE
                       END
```

Es ergaben sich folgende Ergebnisse für den Fall Cv = 0:

ADAM - EVA		ERG	BED	--HI NZVC	Bemerkung
02	01	01	D1	1100 0001	2-1 = 1
09	09	00	D5	1100 0101	9-9 = 0
99	01	98	FB	1111 1011	99-1 = 98
99	99	00	D5	1100 0101	99-99 = 0
01	02	99	D8	1100 1000	1-2 = Unterlauf
10	11	99	D8	1100 1000	10-11 = Unterlauf
AA	01	A9	FB	1111 1011	keine BCD-Ziffer

Wegen der Komplementierung dreht sich nun die Bedeutung des letzten Übertrags Cn um.

Cn = 1 : Ergebnis im zulässigen Bereich
Cn = 0 : Zahlenunterlauf bzw. Borgen bei der nächsten Subtraktion.

6.3.3 Die arithmetischen Befehle des 8080/8085 **8080**

Alle arithmetischen Befehle arbeiten in der Dualarithmetik; für BCD-codierte Dezimalzahlen ist nachträglich eine Korrektur mit dem Befehl DAA erforderlich. Das H-Bit (Halb-Übertrag) des

Bedingungsregisters wird 1 gesetzt, wenn ein Übertrag von Bit 3 nach Bit 4 auftritt. Dies ist nur für die Addition von BCD-Zahlen von Bedeutung.
Bei den arithmetischen Befehlen, die das Carry-Bit berücksichtigen, kann dieses vorher mit einem der beiden folgenden Befehle auf 1 gesetzt oder negiert werden.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-PC	
STC		code-eigen	37	1	4	1	Setze Carry-Bit auf 1
CMC		code-eigen	3F	1	4	C	Negiere Carry-Bit

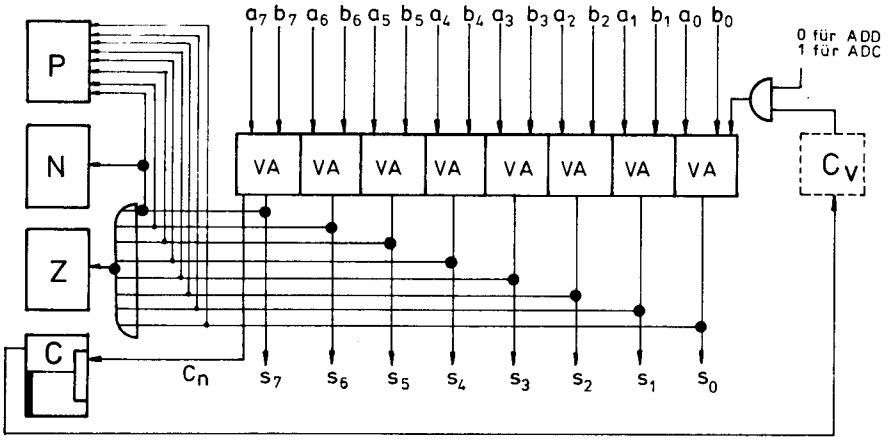
STC bedeutet SeT Carry = setze Carry-Bit
CMC bedeutet CoMplement Carry = komplementiere Carry-Bit

Der Befehl
ORA A setzt das Carry-Bit auf 0, der Befehl
XRA A setzt das Carry-Bit auf 0 und löscht den Akkumulator.

Das folgende Testprogramm untersucht die arithmetischen Befehle:

```
   Hexadezimal          Symbolisch
Adresse Inhalt  Name    Befehl   Operand
                        ORG      2000H
2000    XX      ADAM    DB       XXH
2001    YY      EVA     DB       YYH
2002            PSW     DS       1
2003            ERG     DS       1

                        ORG      2880H
2880    37      START   STC
2881    3F(00)          CMC              (NOP)
2882    3A              LDA      EVA
        01
        20
2885    47              MOV      B,A
2886    3A              LDA      ADAM
        00
        20
2889    88(98)          ADC      B       (SBB)
288A    F5              PUSH     PSW
288B    E1              POP      H
288C    22              SHLD     PSW
        02
        20
288F    C3      OTTO    JMP      OTTO
        8F
        28
                        END
```

Für die Additionsbefehle gilt folgendes Modell:



Das Z-Bit prüft das Ergebnis auf Null. Sind alle acht Bits Null, so ist Z = 1 (ja).
Ist nur ein Bit ungleich Null, so ist Z = 0 (nein).
Das P-Bit zeigt die Parität des Ergebnisses an. Ist die Summe aller 1er-Bits eine gerade Zahl (0,2,4 usw.), so ist P = 1; ist die Summe aller 1er-Bits ungerade (1,3,5 usw.), so ist P = 0.
Die Anzeige dient zur Kontrolle von Datenübertragungen.
Für den Befehl ADD (Addiere ohne Berücksichtigung des Carry-Bits) ist das Carry-Bit vom Eingang des wertniedrigsten Volladdierers getrennt, der Eingang ist konstant 0.

S = A + B

Die folgenden Befehle addieren zum Inhalt des Akkumulators einen zweiten Operanden ohne Berücksichtigung des Carry-Bits.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
ADI	konst	unmittelbar	C6	2	7	xx x x x	Akku ← Akku + Konstante
ADD	A	code-eigen	87	1	4	xx x x x	Akku ← Akku + Akku
ADD	B	code-eigen	80	1	4	xx x x x	Akku ← Akku + B-Register
ADD	C	code-eigen	81	1	4	xx x x x	Akku ← Akku + C-Register
ADD	D	code-eigen	82	1	4	xx x x x	Akku ← Akku + D-Register
ADD	E	code-eigen	83	1	4	xx x x x	Akku ← Akku + E-Register
ADD	H	code-eigen	84	1	4	xx x x x	Akku ← Akku + H-Register
ADD	L	code-eigen	85	1	4	xx x x x	Akku ← Akku + L-Register
ADD	M	Registerpaar	86	1	7	xx x x x	Akku ← Akku + Speicher (Adresse in H,L)

Für den Befehl ADC (Addiere mit Carry-Bit) wird der Inhalt des Carry-Bits auf den Eingang des wertniedrigsten Volladdierers gegeben und bei der Addition mit berücksichtigt.

S = A + B + C

Die folgenden Befehle addieren zum Inhalt des Akkumulators das Carry-Bit und einen zweiten Operanden.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
ACI	konst	unmittelbar	CE	2	7	xx x x x	Akku ← Akku + C + Konstante
ADC	A	code-eigen	8F	1	4	xx x x x	Akku ← Akku + C + Akku
ADC	B	code-eigen	88	1	4	xx x x x	Akku ← Akku + C + B-Register
ADC	C	code-eigen	89	1	4	xx x x x	Akku ← Akku + C + C-Register
ADC	D	code-eigen	8A	1	4	xx x x x	Akku ← Akku + C + D-Register
ADC	E	code-eigen	8B	1	4	xx x x x	Akku ← Akku + C + E-Register
ADC	H	code-eigen	8C	1	4	xx x x x	Akku ← Akku + C + H-Register
ADC	L	code-eigen	8D	1	4	xx x x x	Akku ← Akku + C + L-Register
ADC	M	Registerpaar	8E	1	7	xx x x x	Akku ← Akku + C + Speicher (Adresse in H,L)

Die folgenden Befehle addieren zum Inhalt des Registerpaares H,L den Inhalt eines zweiten Registerpaares; sie führen also eine 16-Bit-Addition aus. Das Carry-Bit wird nicht berücksichtigt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C	
DAD	B	code-eigen	09	1	10	x	H,L-Register ← H,L-Register + B,C-Register
DAD	D	code-eigen	19	1	10	x	H,L-Register ← H,L-Register + D,E-Register
DAD	H	code-eigen	29	1	10	x	H,L-Register ← H,L-Register + H,L-Register
DAD	SP	code-eigen	39	1	10	x	H,L-Register ← H,L-Register + Stapelzeiger

Der DAD-Befehl ermöglicht das Laden des Stapelzeigers in das H,L-Register und damit das Retten in den Variablenspeicher.

Beispiel:

Der Inhalt des Stapelzeigers ist in die Speicherstelle RETT zu bringen.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2000H
2000            RETT   DS       2

                       ORG      2880H
2880      21           LXI      H,00
          00
          00
2883      39           DAD      SP
2884      22           SHLD     RETT
          00
          20
```

a.) Addieren positiver Dualzahlen

Der Zahlenbereich erstreckt sich von 0 bis +255.
Der Carry-Ausgang des werthöchsten Volladdierers ergibt den neuen Inhalt des Carry-Bits (Cn).

Cn = 0 : Das Ergebnis ist kleiner als 256, kein Überlauf
Cn = 1 : Das Ergebnis ist größer als 255, Überlauf

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	+ EVA	+ C	= SUM	PSW	NZ-H -P-C	Bemerkung
01	01	0	02	00	0000 0000	1+1+0=2
FF	01	0	00	55	0101 0101	255+1+0=256 Überlauf
01	01	1	03	04	0000 0100	1+1+1=3
FF	01	1	01	11	0001 0001	255+1+1=257 Überlauf
FE	01	1	00	55	0101 0101	254+1+1=256 Überlauf

b.) Addition vorzeichenbehafteter Dualzahlen

Der Zahlenbereich erstreckt sich von -128 bis +127.
Das Carry-Bit hat hier keine Bedeutung, wird aber beim Befehl ADC (Addiere mit Carry) mit berücksichtigt.
Das N-Bit gibt das Vorzeichen des Ergebnisses an.

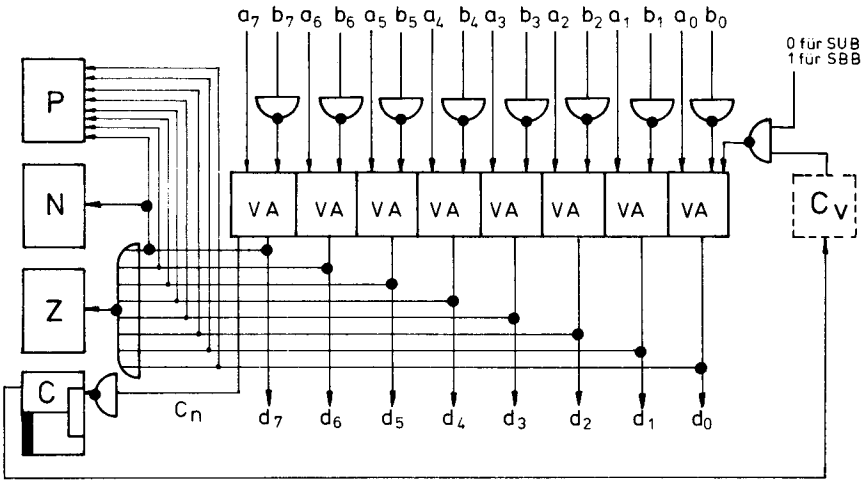
N=0: Ergebnis positiv.
N=1: Ergebnis negativ.

Die Mikroprozessoren 8080 und 8085 haben kein V-Bit, um den Überlauf bzw. Unterlauf vorzeichenbehafteter Zahlen zu prüfen. Sollte dies erforderlich sein, so müsste das Programm die Vorzeichen der beiden Operanden und des Ergebnisses vergleichen.Bei Testläufen mit einem 8085 zeigte es sich, daß das nicht benutzte Bit 1 des Bedingungsregisters den Überlauf (V-Bit) anzeigte.

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	+ EVA	+ C	= SUM	PSW	NZ-H	-P-C	Bemerkung
01	FF	0	00	55	0101	0101	1+(-1)=0
FF	01	0	00	55	0101	0101	(-1)+1=0
02	FD	0	FF	A4	1010	0100	2+(-3)=-1
FD	02	0	FF	A4	1010	0100	(-3)+2=-1
7F	01	0	80	92	1001	0010	127+1=+128 Überlauf
80	FF	0	7F	23	0010	0011	-128+(-1)=-129 Unterlauf
7F	81	0	00	55	0101	0101	127+(-127)=0
81	7F	0	00	55	0101	0101	-127+(+127)=0

Für die Subtraktion gilt folgendes Modell:



Die Subtraktion wird auf eine Addition des 2er-Komplementes zurückgeführt. Die Negationsschaltungen am Eingang der acht Volladdierer bilden das Einerkomplement. Zusätzlich muß über den Cv-Eingang des wertniedrigsten Volladdierers eine 1 addiert werden. Da sich bei jeder Subtraktion die Bedeutung des Carry-Bits umdreht, negieren die Mikroprozessoren 8080 und 8085 automatisch bei jeder Subtraktion das Carry-Bit; sowohl am Eingang des wertniedrigsten Volladdierers als auch am Ausgang des werthöchsten.

Für den Befehl SUB (Subtrahiere ohne Carry) liegt am Eingang des wertniedrigsten Volladdierers immer eine 1.

$$D = A - B$$

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ	H	P	C	
SUI	konst	unmittelbar	D6	2	7	XX	X	X	X	Akku ← Akku - Konstante
SUB	A	code-eigen	97	1	4	XX	X	X	X	Akku ← Akku - Akku
SUB	B	code-eigen	90	1	4	XX	X	X	X	Akku ← Akku - B-Register
SUB	C	code-eigen	91	1	4	XX	X	X	X	Akku ← Akku - C-Register
SUB	D	code-eigen	92	1	4	XX	X	X	X	Akku ← Akku - D-Register
SUB	E	code-eigen	93	1	4	XX	X	X	X	Akku ← Akku - E-Register
SUB	H	code-eigen	94	1	4	XX	X	X	X	Akku ← Akku - H-Register
SUB	L	code-eigen	95	1	4	XX	X	X	X	Akku ← Akku - L-Register
SUB	M	Registerpaar	96	1	7	XX	X	X	X	Akku ← Akku - Speicher (Adresse in H,L)

Für den Befehl SBB (Subtrahiere mit Borgen) wird das negierte Carry-Bit = Borrow (Borgen) auf den Carry-Eingang des wertniedrigsten Volladdierers gegeben. Der SBB-Befehl dient zur gruppenweisen Subtraktion von Dualzahlen, die mehr als acht bit lang sind.
Ist bei einer Gruppe der Subtrahend (B) größer als der Minuend (A), so muß bei der folgenden Subtraktion zusätzlich eine 1 abgezogen werden (Borrow = Borgen); die Addition der 1 zur Komplementbildung unterbleibt.

$$D = A - B - 1$$

Die folgenden Befehle subtrahieren vom Inhalt des Akkumulators das Carry-Bit und einen zweiten Operanden.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ	H	P	C	
SBI	konst	unmittelbar	DE	2	7	XX	X	X	X	Akku ← Akku - C - Konstante
SBB	A	code-eigen	9F	1	4	XX	X	X	X	Akku ← Akku - C - Akku
SBB	B	code-eigen	98	1	4	XX	X	X	X	Akku ← Akku - C - B-Register
SBB	C	code-eigen	99	1	4	XX	X	X	X	Akku ← Akku - C - C-Register
SBB	D	code-eigen	9A	1	4	XX	X	X	X	Akku ← Akku - C - D-Register
SBB	E	code-eigen	9B	1	4	XX	X	X	X	Akku ← Akku - C - E-Register
SBB	H	code-eigen	9C	1	4	XX	X	X	X	Akku ← Akku - C - H-Register
SBB	L	code-eigen	9D	1	4	XX	X	X	X	Akku ← Akku - C - L-Register
SBB	M	Registerpaar	9E	1	7	XX	X	X	X	Akku ← Akku - C - Speicher (Adresse in H,L)

SBB bedeutet SUBtrahiere mit Borgen.

a.) Subtrahieren positiver Dualzahlen

Der Zahlenbereich erstreckt sich von 0 bis +255. Ergibt sich ein negativer Wert, so spricht man von einem Zahlenunterlauf. Da am Ausgang des werthöchsten Volladdierers bei jeder Subtraktion das Carry-Bit negiert wird, hat es die gleiche Bedeutung wie bei der Addition.

Cn = 0 : Das Ergebnis ist größer oder gleich 0, kein Zahlenunterlauf.
Cn = 1 : Das Ergebnis ist kleiner 0, Zahlenunterlauf.

Das Testprogramm lieferte folgende Ergebnisse:

ADAM	- EVA	- C	= ERG	PSW	NZ-H -P-C	Bemerkung
02	01	0	01	10	0001 0000	2-1-0=1
01	01	0	00	54	0101 0100	1-1-0=0
01	02	0	FF	A5	1010 0101	1-2-0=-1 Unterlauf
03	01	1	01	10	0001 0000	3-1-1=1
02	01	1	00	54	0101 0100	2-1-1=0
01	01	1	FF	A5	1010 0101	1-1-1=-1 Unterlauf

b.) Subtrahieren vorzeichenbehafteter Dualzahlen

Der Zahlenbereich erstreckt sich von -128 bis +127. Der Ausgang Cn des werthöchsten Volladdierers hat in dieser Zahlendarstellung keine Bedeutung. Das N-Bit zeigt das Vorzeichen des Ergebnisses an.

N = 0 : Ergebnis positiv
N = 1 : Ergebnis negativ

Die Mikroprozessoren 8080 und 8085 haben kein V-Bit. Daher ist es nach der Subtraktion vorzeichenbehafteter Dualzahlen nicht möglich, Überlauf bzw. Unterlauf zu prüfen.

Das Testprogramm lieferte für C = 0 folgende Ergebnisse:

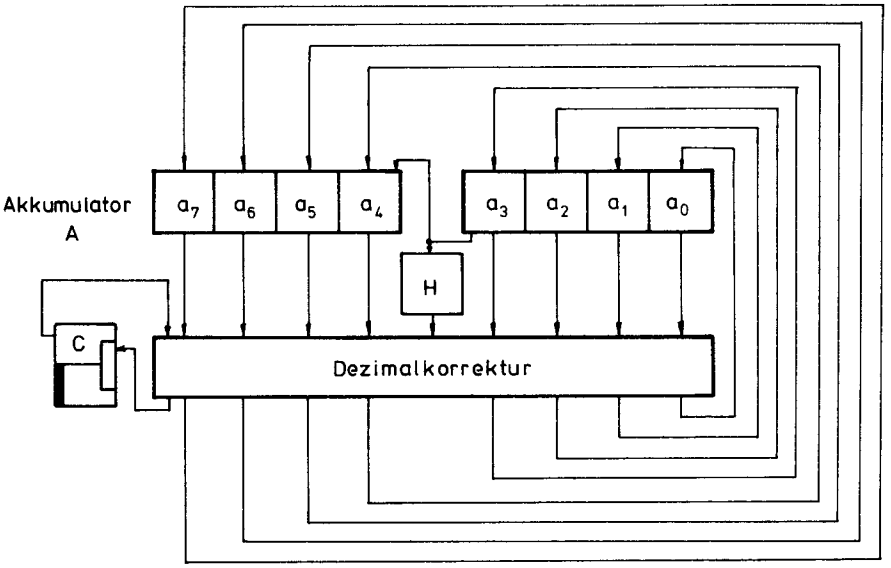
ADAM	- EVA	= ERG	PSW	NZ-H -P-C	Bemerkung
01	02	FF	A5	1010 0101	(+1)-(+2)=-1
FE	FF	FF	A5	1010 0101	(-2)-(-1)=-1
FF	FE	01	10	0001 0000	(-1)-(-2)=+1
FF	FF	00	54	0101 0100	(-1)-(-1)=0
7F	FF	80	93	1001 0011	(+127)-(-1)=128 Überlauf
80	01	7F	22	0010 0010	(-128)-(+1)=-129 Unterlauf
80	FF	81	A5	1010 0101	(-128)-(-1)=-127
02	FF	03	05	0000 0101	(+2)-(-1)=3
00	00	00	54	0101 0100	0-0=0

Beim Rechnen mit BCD-codierten Dezimalzahlen muß nach jedem Additionsbefehl im Akkumulator eine Dezimalkorrektur vorgenommen werden. Nach einem Subtraktionsbefehl ist keine Dezimalkorrektur möglich, da das Hilfs-Carry-Bit nicht negiert wird.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N Z H P C	
DAA		code-eigen	27	1	4	X X X X X	Dezimalkorrektur im Akku für BCD-Zahlen

DAA bedeutet Decimal Adjust Accumulator = Dezimalkorrektur

Für den Befehl gilt folgendes Modell:



Bei der Arbeit mit BCD-codierten Dezimalzahlen (8-4-2-1-Code) enthält ein Byte zwei positive Dezimalziffern. In den Bitpositionen a_0 bis a_3 (rechtes Halbbyte) steht die wertniedrigere Ziffer, in den Bitpositionen a_4 bis a_7 (linkes Halbbyte) die werthöhere. Das H-Bit, das den Übertrag von Bitposition 3 nach Bitposition 4 anzeigt, wird nur durch die Additionsbefehle so beeinflußt, daß der Befehl DAA das Ergebnis richtig korrigieren kann.

Der Zahlenbereich positiver Dezimalzahlen erstreckt sich von 00 bis +99. Auch hier wird unter Berücksichtigung des Vorübertrags Cv addiert. Ist der Ausgang des werthöchsten Volladdierers Cn=1, so ist das Ergebnis größer als 99. Verarbeitet man mehrstellige Zahlen, so muß der Übertrag auf die nächste Stelle addiert werden (Dekadenübertrag). Das folgende Testprogramm addiert zwei zweistellige BCD-Zahlen.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2000H
2000      XX    ADAM   DB       XXH
2001      YY    EVA    DB       YYH
2002            PSW    DS       1
2003            SUM    DS       1

                       ORG      2880H
2880      B7    START  ORA      A
2881      3A           LDA      EVA
          01
          20
2884      4F           MOV      C,A
2885      3A           LDA      ADAM
          00
          20
2888      81           ADD      C
2889      27           DAA
288A      F5           PUSH     PSW
288B      E1           POP      H
288C      22           SHLD     PSW
          02
          20
288F      C3    OTTO   JMP      OTTO
          8F
          28
                       END
```

Es lieferte folgende Ergebnisse für den Fall Cv=0:

ADAM	+ EVA	= SUM	PSW	NZ-H -P-C	Bemerkung
02	01	03	04	0000 0100	2+1=3
09	09	18	04	0000 0100	9+9=18
99	01	00	55	0101 0101	99+1=100 Überlauf
99	99	98	83	1000 0011	99+99=198 Überlauf
01	02	03	04	0000 0100	1+2=3
10	11	21	04	0000 0100	10+11=21
AA	01	11	15	0001 0101	keine BCD-Ziffer

Bei der Subtraktion positiver BCD-Zahlen muß im Programm das 10er-Komplement gebildet und addiert werden.
Dazu ist zunächst die Differenz zur Konstanten 9 zu bilden (9er-Komplement) und bei der nachfolgenden Addition zusätzlich eine 1 zu addieren (Cv=1). Durch Vernachlässigung des Übertrags wird die Komplementbildung rückgängig gemacht.

Zahlenbeispiel:

8 - 3 => 8 + (9-3) + 1 => 8 + 6 + 1 => 15 - 10 => 5

Das folgende Testprogramm untersucht dieses Verfahren für zweistellige BCD-Zahlen.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2000H
2000      XX    ADAM   DB       XXH
2001      YY    EVA    DB       YYH
2002            PSW    DS       1
2003            DIFF   DS       1

                       ORG      2880H
2880      3A    START  LDA      EVA
          01
          28
2883      4F           MOV      C,A
2884      3A           LDA      ADAM
          00
          20
2887      47           MOV      B,A
2888      3E           MVI      A,99H
          99
288A      91           SUB      C
288B      37           STC
288C      88           ADC      B
288D      27           DAA
288E      F5           PUSH     PSW
288F      E1           POP      H
2890      22           SHLD     PSW
          02
          20
2893      C3    OTTO   JMP      OTTO
          93
          28
                       END
```

Es ergaben sich folgende Ergebnisse für den Fall Cv=0:

ADAM	- EVA	= DIFF	PSW	NZ-H -P-C	Bemerkung
02	01	01	11	0001 0001	2-1=1
09	09	00	55	0101 0101	9-9=0
99	01	98	83	1000 0011	99-1=98
99	99	00	55	0101 0101	99-99=0
01	02	99	A4	1010 0100	1-2=-1 Unterlauf
10	21	89	A0	1010 0000	10-21=-1 Unterlauf
AA	01	A9	87	1000 0111	keine BCD-Zahl

Wegen der Komplementbildung dreht sich nun die Bedeutung des letzten Übertrags Cn um:

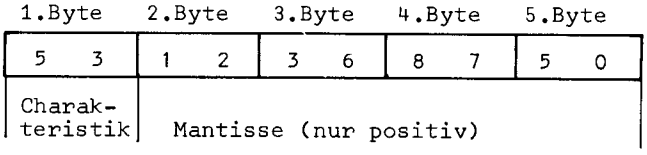
Cn = 1 : Ergebnis im zulässigen Bereich
Cn = 0 : Zahlenunterlauf bzw. Borgen bei der nächsten Subtraktion.

6.4 Erweiterung des Zahlenbereiches

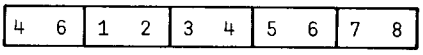
Die Akkumulatoren der hier behandelten Mikroprozessoren sind acht bit lang. Sie verarbeiten ein Byte = acht Bits gleichzeitig (parallel). Längere Zahlen müssen in Gruppen zu je einem Byte nacheinander (seriell) behandelt werden. Bei 16-Bit-Operationen kann man noch zwei Register zusammenschalten, bei längeren Operanden arbeitet man zweckmäßigerweise mit Schleifen.

Ein besonderes Problem ist die Darstellung reeller Zahlen, also von Zahlen mit Stellen hinter dem Komma. Grundsätzlich unterscheidet man zwischen BCD-codierten Dezimalzahlen und Dualzahlen. In beiden Fällen werden die Zahlen wie in der Mathematik und Technik gebräuchlich mit Mantisse und Exponent geschrieben; also für 123,6875 setzt man $0{,}1236875 \cdot 10^3$

1.Beispiel: Positive BCD-Zahlen in fünf Bytes.
Die Zahl und damit die Mantisse sei immer positiv. Um auch negative Exponenten darstellen zu können, addiert man zu jedem Exponenten die Konstante 50 und erfaßt mit zwei Dezimalstellen einen Zahlenbereich von 10^{-50} bis 10^{+49}. Die zweistellige Charakteristik ergibt sich aus dem Exponenten plus Konstante 50. Damit wird die Zahl $0{,}1236875 \cdot 10^3$ in folgender Form gespeichert:



Die Zahl $0{,}000012345678 = 0{,}12345678 \cdot 10^{-4}$ ergibt sich zu



Der Zahlenbereich liegt zwischen 10^{-50} bis 10^{+49} bei einer Genauigkeit von 8 Dezimalstellen. Zur Kennzeichnung negativer Zahlen könnte man eine Marke an das Ende der Zahl anfügen, so z.B. 0 für positiv und F für negativ.

2.Beispiel: Positive Dualzahlen in fünf Bytes

Die Zahl wird als reine Dualzahl dargestellt und so normalisiert, daß das Dualkomma vor der werthöchsten Stelle steht.

$$123{,}6875_{10} = 1111011{,}1011_2 = 0{,}11110111011_2 \cdot 2^7$$

Zum Exponenten $7_{10} = 111_2$ wird die Konstante 10000000 addiert, um auch negative Exponenten speichern zu können. Damit ergibt sich die Charakteristik $111_2 + 10000000_2 = 10000111_2$.
Die Zahl besteht nun aus einer 8 bit langen Charakteristik und einer 32 bit langen Mantisse. Damit wird die Zahl 123,6875 in folgender Form gespeichert

1.Byte	2.Byte	3.Byte	4.Byte	5.Byte
1000 0111	1111 0111	0110 0000	0000 0000	0000 0000
Charakteristik	Mantisse (nur positiv)			

, Dualkomma

Der Zahlenbereich liegt zwischen 10^{-38} bis 10^{+37} bei einer Genauigkeit von 9 Dezimalstellen. Zur Darstellung negativer Zahlen könnte man die Mantisse im Komplement darstellen.

Für die Verarbeitung der in den beiden Beispielen genannten Zahlendarstellungen wird in der Praxis ein System von Unterprogrammen aufgebaut, das oft auch Programme für das Wurzelziehen und zur Berechnung von logarithmischen und trigonometrischen Funktionen enthält. Das Problem kann aber auch hardwaremäßig über Rechenbausteine gelöst werden.

Die folgenden Abschnitte enthalten einige Beispiele für das Rechnen mit erhöhter Genauigkeit.

6.4.1 Beispiele für den KIM-1 KIM

a.) Rechnen mit 16 bit langen positiven Dualzahlen

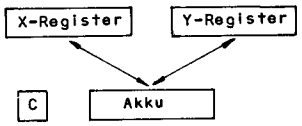
Als Beispiel diene die Formel

$$ERG = (OP1 + OP2) \cdot 2$$

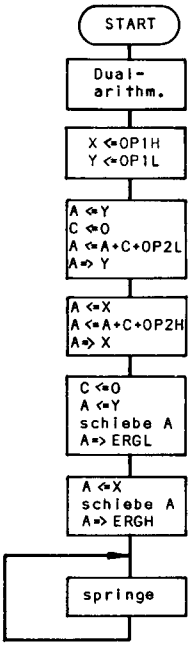
Die Operationen

Laden einer 16-Bit-Größe,
Addieren einer 16-Bit-Größe,
Schieben einer 16-Bit-Größe (Multiplizieren mit 2) und
Speichern einer 16-Bit-Größe

sollen ohne Hilfsspeicherstellen nur mit den Registern des Mikroprozessors ausgeführt werden. Da der Akkumulator nur acht bit lang ist, dienen die Register X und Y als Zwischenspeicher.



Das folgende Testprogramm berechnet die Formel für die dezimalen Werte (4711 + 1000)·2 = 11422 oder hexadezimal (1267 + 03E8)·2= 2C9E.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0000	
0000	12	OP1H	.BYTE	$12
0001	67	OP1L	.BYTE	$67
0002	03	OP2H	.BYTE	$03
0003	E8	OP2L	.BYTE	$E8
0004		ERGH	*=*+1	
0005		ERGL	*=*+1	
			*=$0300	
0300	D8	START	CLD	
0301	AE		LDX	OP1H
	00			
	00			
0304	AC		LDY	OP1L
	01			
	00			
0307	98		TYA	
0308	18		CLC	
0309	6D		ADC	OP2L
	03			
	00			
030C	A8		TAY	
030D	8A		TXA	
030E	6D		ADC	OP2H
	02			
	00			
0311	AA		TAX	
0312	98		TYA	
0313	18		CLC	
0314	2A		ROLA	
0315	8D		STA	ERGL
	05			
	00			
0318	8A		TXA	
0319	2A		ROLA	
031A	8D		STA	ERGH
	04			
	00			
031D	4C	OTTO	JMP	OTTO
	1D			
	03			
			.END	

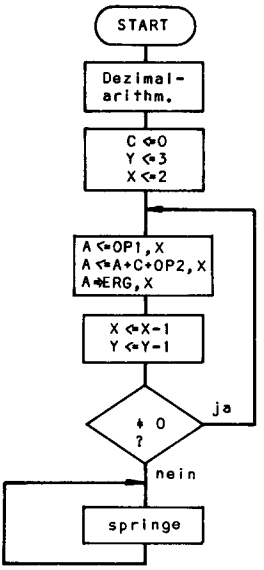
Das X-Register speichert den niederwertigen Teil des Ergebnisses, das Y-Register den höherwertigen Teil. Das Carry-Bit verbindet beim Addieren und Schieben die beiden Teile miteinander. Da bei noch längeren Zahlen die Register des Mikroprozessors nicht ausreichen, muß im RAM-Bereich ein entsprechend langer Hilfs-Akkumulator geschaffen werden.

b.) Rechnen mit sechsstelligen positiven BCD-Zahlen

Als Beispiel diene die Formel

$$ERG = OP1 + OP2$$

Die Operanden sind in jeweils drei Bytes gespeichert. Alle drei Bereiche werden in einer Schleife mit Hilfe des X-Registers adressiert, das Y-Register zählt die Schleifendurchläufe. Eine schnellere und elegantere Lösung würde das X-Register zum Addressieren und Zählen benutzen. Das folgende Testprogramm berechnet die Formel für die Werte 4711 + 1000 = 5711.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0000	
0000	00	OP1	.BYTE	$00
0001	47		.BYTE	$47
0002	11		.BYTE	$11
0003	00	OP2	.Byte	$00
0004	10		.BYTE	$10
0005	00		.BYTE	$00
0006		ERG	*=*+3	
			*=$0300	
0300	F8	START	SED	
0301	18		CLC	
0302	A0		LDY	#$03
	03			
0304	A2		LDX	#$02
	02			
0306	BD	EVA	LDA	OP1,X
	00			
	00			
0309	7D		ADC	OP2,X
	03			
	00			
030C	9D		STA	ERG,X
	06			
	00			
030F	CA		DEX	
0310	88		DEY	
0311	D0		BNE	EVA
	F3			
0313	4C	OTTO	JMP	OTTO
	13			
	03			
			.END	

Für mehrere Operationen müßte ein Hilfs-Akkumulator entsprechender Länge im RAM-Bereich errichtet werden, der die Zwischenergebnisse aufnimmt.

6.4.2 Beispiele für den 6800

6800

a.) Rechnen mit 16 bit langen positiven Dualzahlen

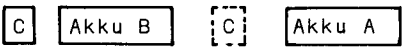
Als Beispiel diene die Formel

$$ERG = (OP1 + OP2) \cdot 2$$

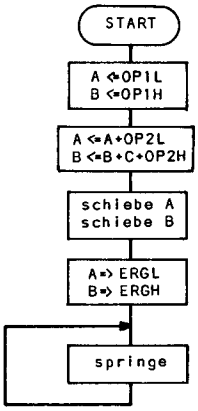
Die Operationen

Laden einer 16-Bit-Größe,
Addieren einer 16-Bit-Größe,
Schieben einer 16-Bit-Größe (Multiplizieren mit 2) und
Speichern einer 16-Bit-Größe

sollen ohne Hilfsspeicherstellen nur mit den Registern des Mikroprozessors ausgeführt werden. Die beiden 16 bit langen Register (Indexregister und Stapelzeiger) erlauben weder arithmetische noch logische Operationen oder Verschiebungen und haben auch keine direkte Verbindung zu den beiden Akkumulatoren, die sich jedoch mit Hilfe des Carry-Bits zu einem 16-Bit-Register zusammenschalten lassen.



Das folgende Testprogramm berechnet die Formel für die dezimalen Werte (4711 + 1000) · 2 = 11422 oder hexadezimal (1267 + 03E8) · 2 = 2C9E.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0000
0000	12	OP1H	FCB	$12
0001	67	OP1L	FCB	$67
0002	03	OP2H	FCB	$03
0003	E8	OP2L	FCB	$E8
0004		ERGH	RMB	1
0005		ERGL	RMB	1
			ORG	$0180
0180	B6	START	LDA A	OP1L
	00			
	01			
0183	F6		LDA B	OP1H
	00			
	00			
0186	BB		ADD A	OP2L
	00			
	03			
0189	F9		ADC B	OP2H
	00			
	02			
018C	48		ASL A	
018D	59		ROL B	
018E	B7		STA A	ERGL
	00			
	05			
0191	F7		STA B	ERGH
	00			
	04			
0194	20	OTTO	BRA	OTTO
	FE			
			END	

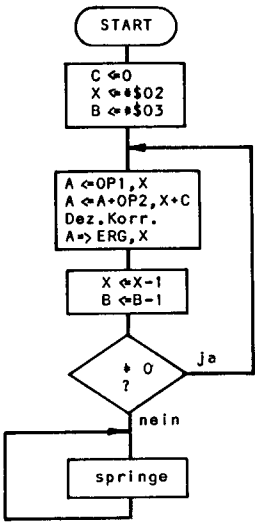
Bei noch längeren Zahlen und Rechnungen mit Zwischenergebnissen müßte ein Hilfs-Akkumulator im RAM-Bereich errichtet werden.

b.) Rechnen mit sechsstelligen positiven BCD-Zahlen

Als Beispiel diene die Formel

ERG = OP1 + OP2

Die Operanden sind in jeweils drei Bytes gespeichert und liegen zusammen in Seite Null. Der Akkumulator B zählt die Schleifendurchläufe. Die indizierten Befehle enthalten im zweiten Byte die Anfangsadresse des jeweiligen Bereiches, im Indexregister steht der gemeinsame veränderliche Abstand. Eine schnellere und elegantere Lösung würde das Indexregister zum Zählen und Indizieren benutzen. Das folgende Testprogramm berechnet die Formel für die Werte 4711 + 1000 = 5711.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0000
0000	00	OP1	FCB	$00
0001	47		FCB	$47
0002	11		FCB	$11
0003	00	OP2	FCB	$00
0004	10		FCB	$10
0005	00		FCB	$00
0006		ERG	RMB	3
			ORG	$0180
0180	0C	START	CLC	
0181	CE		LDX	#$0002
	00			
	02			
0184	C6		LDA B	#$03
	03			
0186	A6	BUBI	LDA A	OP1,X
	00			
0188	A9		ADC A	OP2,X
	03			
018A	19		DAA	
018B	A7		STA A	ERG,X
	06			
018D	09		DEX	
018E	5A		DEC B	
018F	26		BNE	BUBI
	F5			
0191	20	OTTO	BRA	OTTO
	FE			
			END	

Für mehrere Operationen müßte ein Hilfs-Akkumulator entsprechender Länge im RAM-Bereich errichtet werden, der die Zwischenergebnisse aufnimmt.

6.4.3 Beispiele für den 8080/8085 **8080**

a.) Rechnen mit 16 bit langen positiven Dualzahlen

Als Beispiel diene die Formel

$$ERG = (OP1 + OP2) \cdot 2$$

Die Operationen

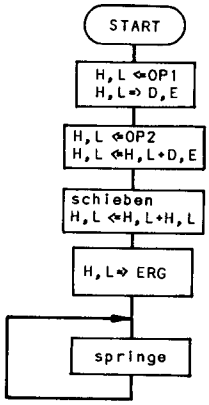
Laden einer 16-Bit-Größe,
Addieren einer 16-Bit-Größe,
Schieben einer 16-Bit-Größe (Multiplizieren mit 2) und
Speichern einer 16-Bit-Größe

können direkt mit dem H,L-Register des Mikroprozessors vorgenommen werden. Als Hilfsspeicher dienen die beiden anderen Registerpaare B,C und D,E. Das Registerpaar H,L ist ein 16-Bit-Akkumulator mit folgendem Befehlssatz:

LHLD	Adresse	Laden mit einer 16-Bit-Speicherstelle.
SHLD	Adresse	Speichern in eine 16-Bit-Speicherstelle.
XCHG		Vertausche die Inhalte der Registerpaare H,L und D,E
DAD	Registerpaar	Addiere H,L-Register mit anderem Registerpaar; das Carry-Bit wird beeinflußt.
DAD	H,L	Addiere H,L mit sich selbst, d.h. verschiebe es um 1 Bit nach links; das Carry-Bit nimmt die werthöchste Stelle auf.

Dazu kommen die 16-Bit-Befehle LXI, INX, DCX, PUSH, POP, XTHL, PCHL und SPHL.

Das folgende Testprogramm berechnet die Formel für die dezimalen Werte (4711 + 1000) * 2 = 11422 oder hexadezimal (1267 + 03E8) * 2 = 2C9E.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000	67	OP1	DW	1267H
	12			
2002	E8	OP2	DW	03E8H
	03			
2004		ERG	DS	2
			ORG	2880H
2880	2A	START	LHLD	OP1
	00			
	20			
2883	EB		XCHG	
2884	2A		LHLD	OP2
	02			
	20			
2887	19		DAD	D
2888	29		DAD	H
2889	22		SHLD	ERG
	04			
	20			
288C	C3	OTTO	JMP	OTTO
	8C			
	28			
			END	

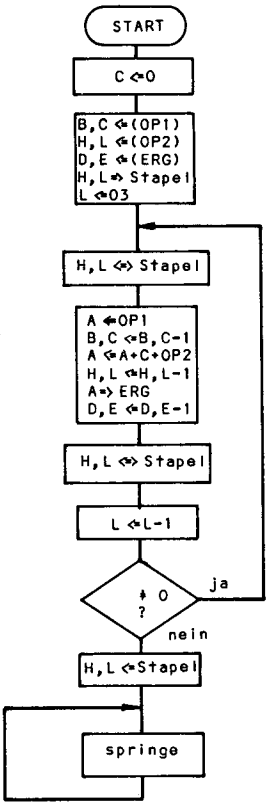
b.) Rechnen mit sechsstelligen positiven BCD-Zahlen

Als Beispiel diene die Formel

ERG = OP1 + OP2

Die Operanden sind in jeweils drei Bytes gespeichert. In der Verarbeitungsschleife indiziert das Registerpaar B,C den Bereich OP1, das Registerpaar H,L den Bereich OP2 und das Paar D,E das Ergebnis ERG. Zur Kontrolle der Schleifendurchläufe dient ein Zähler im L-Register. Da das L-Register sowohl als Zähler als auch zur Indizierung dient, wird das Registerpaar H,L mit dem Befehl XTHL dauernd mit dem obersten 16-Bit-Wert des Stapels vertauscht.

Das folgende Testprogramm berechnet die Formel für die Werte 4711 + 1000 = 5711.



Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2000H
2000	00	OP1	DB	00H
2001	47		DB	47H
2002	11		DB	11H
2003	00	OP2	DB	00H
2004	10		DB	10H
2005	00		DB	00H
2006		ERG	DS	3
			ORG	2880H
2880	B7	START	ORA	A
2881	01		LXI	B,OP1+2
	02			
	20			
2884	21		LXI	H,OP2+2
	05			
	20			
2887	11		LXI	D,ERG+2
	08			
	20			
288B	E5		PUSH	H
288C	2E		MVI	L,03H
	03			
288E	E3	EVA	XTHL	
288F	0A		LDAX	B
2890	0B		DCX	B
2891	8E		ADC	M
2892	27		DAA	
2893	2B		DCX	H
2894	12		STAX	D
2895	1B		DCX	D
2896	E3		XTHL	
2897	2D		DCR	L
2898	C2		JNZ	EVA
	8E			
	28			
289B	E1		POP	H
289C	C3	OTTO	JMP	OTTO
	9C			
	28			
			END	

6.5 Multiplikation und Division A

Beide Rechenarten sind in den Befehlssätzen der hier behandelten Mikroprozessoren nicht enthalten. Alle Beispiele und Überlegungen gelten nur für positive Dualzahlen.

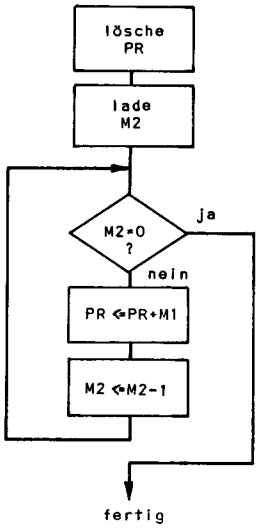
Die Multiplikation

Multiplikand X Multiplikator = Produkt

kann im einfachsten Fall auf eine mehrmalige Addition des Multiplikanden zurückgeführt werden. Mit den Abkürzungen

M1 für den Multiplikanden,
M2 für den Multiplikator und
PR für das Produkt

ergibt sich folgender Ablaufplan:



Dieses Verfahren ist jedoch für den praktischen Gebrauch zu langsam, da die Additionsschleife bei einem großen Multiplikator sehr oft durchlaufen wird. Schneller ist das bei der Handrechnung gebräuchliche Verfahren, das Teilprodukte unter Berücksichtigung der Stellenwertigkeit addiert. Dabei kann man sowohl mit dem werthöchsten als auch mit dem wertniedrigsten Bit des Multiplikators beginnen.

Zahlenbeispiele dual

a. links beginnend

```
0101 · 0101
0000
 0101
  0000
   0101
0011001
```

b. rechts beginnend

```
0101 · 0101
 0101
  0000
   0101
0000
0011001
```

Das folgende Zahlenbeispiel zeigt, daß die Multiplikation auf Additions- und Schiebebefehle zurückgeführt werden kann.

$$0101 \cdot 0101 = 0101 \cdot (0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)$$

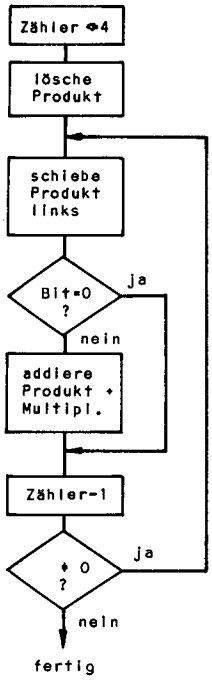
$$= 0101 \cdot 0 \cdot 2^3 + 0101 \cdot 1 \cdot 2^2 + 0101 \cdot 0 \cdot 2^1 + 0101 \cdot 1 \cdot 2^0$$

$$= 0000 \cdot 2^3 + 0101 \cdot 2^2 + 0000 \cdot 2^1 + 0101 \cdot 2^0$$

Zur Multiplikation mit 2^n wird das Teilprodukt um n Stellen nach links geschoben.

Bei der Multiplikation zweier vierstelliger Dualzahlen kann ein achtstelliges Ergebnis entstehen. Der folgende Programmablaufplan beginnt mit dem werthöchsten Bit des Multiplikators. Er benötigt einen Zähler von 1 bis 4, ein 4-Bit-Multiplikanden-Register, ein 8-Bit-Produktregister für die Operationen links Schieben und Addieren und ein 4-Bit-Testregister zur Untersu-

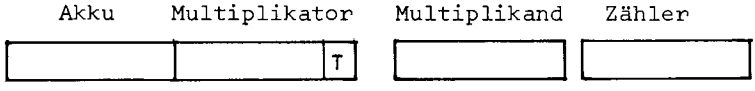
chung des Multiplikators beginnend mit dem werthöchsten Bit. Rechts daneben steht das Zahlenbeispiel
0101 X 0101 = 00011001



0101 = Multiplikand

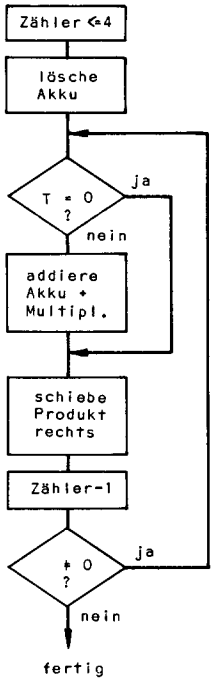
Zähler	Produkt	Multiplikator	Bemerkung
4	00000000	0101	Ausgangszustand
	00000000		Schieben
		0101	Testen
	00000000		nicht addieren
3			Zählen
	00000000		Schieben
	+ 0101	0101	Testen
	00000101		Addieren
2			Zählen
	00001010		Schieben
		0101	Testen
	00001010		nicht addieren
1			Zählen
	00010100		Schieben
	+ 0101	0101	Testen
	00011001		Addieren
0			Zählen
	00011001		Ergebnis

Das folgende Multiplikationsverfahren beginnt mit dem wertniedrigsten Bit des Multiplikators. Es wird in seriellen Rechenwerken zur Multiplikation verwendet; der Akkumulator wird um ein Multiplikator-Quotienten-Register MQR erweitert. Außerdem sind ein Multiplikandenregister und ein Zähler erforderlich.



Für vierstellige Dualzahlen benötigt das Verfahren einen Zähler von 1 bis 4, ein 4-Bit-Multiplikandenregister und ein 8-Bit-Produktregister für die Operationen Rechts-Schieben und Addieren (getrennt für die vier werthöchsten Bits). Das wertniedrigste

Bit des Multiplikators entscheidet über die Addition. Rechts neben dem Programmablaufplan steht das Zahlenbeispiel 0101 X 0101 = 0001 1001.



0101 = Multiplikand

Zähler	Akku	Multiplikator	Bemerkung
4	0000	0101	Ausgangszustand
		1	Testen
	0101	0101	Addieren
	0010	1010	Schieben
3			Zählen
		0	Testen
	0010	1010	nicht addieren
	0001	0101	Schieben
2			Zählen
		1	Testen
	0110	0101	Addieren
	0011	0010	Schieben
1			Zählen
		0	Testen
	0011	0010	nicht addieren
	0001	1001	Schieben
0			Zählen
	0001	1001	Ergebnis

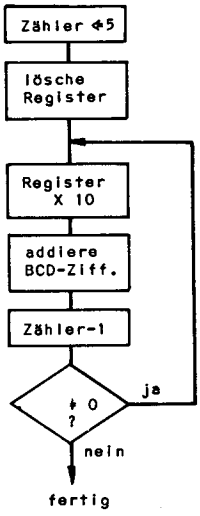
Bei der Umwandlung BCD-codierter Dezimalzahlen in Dualzahlen müssen die Ziffern mit der Konstanten 10 multipliziert werden.

Beispiel: Umwandlung der Dezimalzahl 12345

$$12345 = 1 \cdot 10^4 + 2 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$$

$$= (((\ 1 \cdot 10 + 2\ ) \cdot 10 + 3\ ) \cdot 10 + 4\ ) \cdot 10 + 5$$

Da die Ziffern des BCD-Codes dual verschlüsselt sind, kann man mit der werthöchsten BCD-Ziffer beginnend nach folgendem Ablaufplan vorgehen:



Für Zahlen kleiner 65536 muß das Register 16 bit lang sein. Die Multiplikation mit der Konstanten 10 erfolgt in drei Schritten:

Multiplikanden um zwei bit nach links verschieben (mal 4)
alten Multiplikanden addieren
Summe um ein bit nach links verschieben (mal 2)

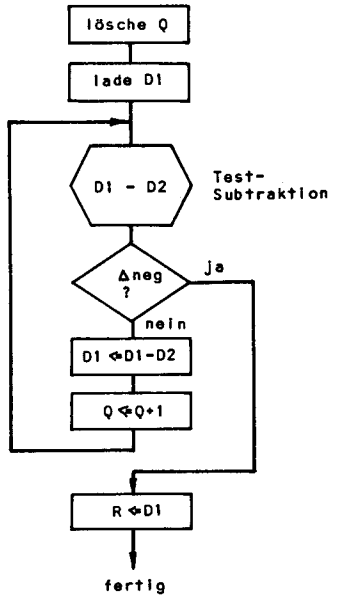
Dies entspricht der Formel

$$(4 \cdot X + X) \cdot 2 = 10\ X$$

Die ganzzahlige Division

Dividend : Divisor = Quotient + Rest

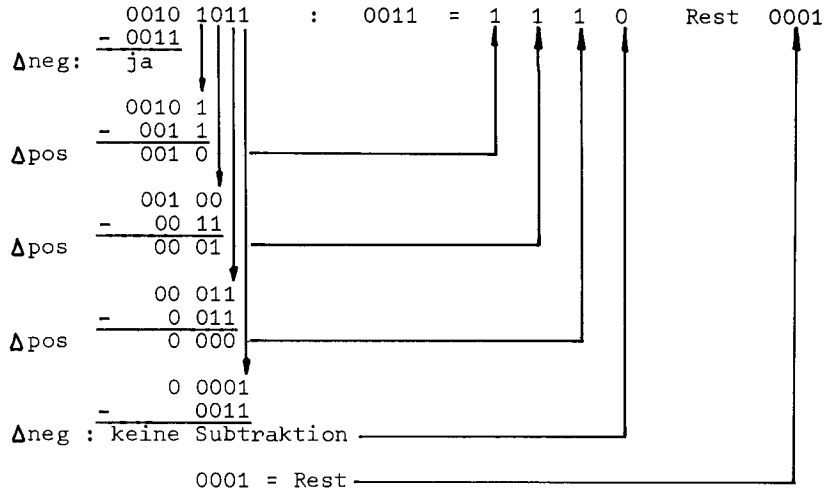
kann im einfachsten Fall auf eine mehrmalige Subtraktion des Divisors vom Dividenden zurückgeführt werden. Mit den Abkürzungen
D1 für den Dividenden,
D2 für den Divisor,
Q für den Quotienten und
R für den Divisionsrest
ergibt sich folgender Ablaufplan:



Der Divisionsrest ergibt sich im Dividendenregister. Die Testsubtraktion läßt sich durch eine echte Subtraktion ersetzen, dann müßte jedoch nach dem Verlassen der Schleife die letzte Subtraktion wieder rückgängig gemacht werden. Bei positiven Dualzahlen muß das Carry-Bit zur Untersuchung der Differenz herangezogen werden. Der Fall Divisor = 0 ist vor der Schleife abzufragen, da diese sonst kein Ende findet.

Das Subtraktionsverfahren ist für den praktischen Gebrauch zu langsam, da die Subtraktionsschleife bei einem kleinen Divisor sehr oft durchlaufen wird. Schneller ist das bei der Handrechnung gebräuchliche Verfahren, das unter Berücksichtigung der Stellenwertigkeit subtrahiert.

Zahlenbeispiel dual



Das Beispiel dividiert einen 8-Bit-Dividenden durch einen 4-Bit-Divisor zu einem 4-Bit-Quotienten und einen 4-Bit-Rest. Die erste Testsubtraktion der vier werthöchsten Dividendenstellen minus Divisor entscheidet, ob die Division ausführbar ist. Ist diese Differenz Null oder positiv, so ist der Divisor Null oder das Ergebnis größer als 4 bit.

Bei allen folgenden Test-Subtraktionen wird jeweils ein anderes Bit des Divisors dazugenommen. Ist die Differenz positiv oder Null, so ergibt sich im Quotienten ein 1er Bit, sonst ein 0er Bit. Das folgende Zahlenbeispiel zeigt, wie sich die Division auf Teilsubtraktionen und Schiebebefehle zurückführen läßt.

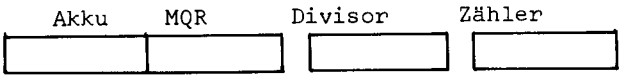
$$00101011 : 0011 = (00101 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) : 0011$$

$$= \frac{00101}{0011} \cdot 2^3 + \frac{2 \cdot Rest + 0}{0011} \cdot 2^2 + \frac{2 \cdot Rest + 1}{0011} \cdot 2^1 + \frac{2 \cdot Rest + 1}{0011} \cdot 2^6 + Rest$$

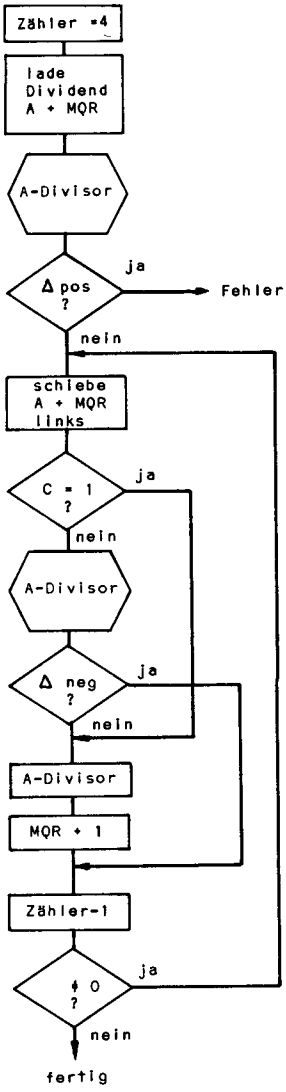
$$= 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \quad \text{Rest } 1$$

Die Teildivisionen werden durch Subtraktionen ersetzt, die Multiplikation des Restes mit 2 durch eine Linksverschiebung.

Das folgende Divisionsverfahren wird in seriellen Rechenwerken verwendet. Dabei wird der Akkumulator um ein Multiplikator-Quotienten-Register MQR erweitert. Außerdem sind ein Divisorregister und ein Zähler erforderlich.



Vor der Division enthalten der Akkumulator und seine Erweiterung den 8 bit langen Dividenden; nach der Division steht der Quotient im MQR und der Rest im Akkumulator. Rechts neben dem Ablaufplan steht das Zahlenbeispiel
00101011 : 0011 = 1110 Rest 0001.



0011 = Divisor

Zähler	C	Akku	MQR	Δ	Bemerkung
4		0010	1011		Ausgangswerte
				neg	Testsubtraktion
	0	0101	0110		Schiebe
				pos	Testsubtraktion
		0010			Subtrahiere
			0111		MQR + 1
3					Zählen
	0	0100	1110		Schieben
				pos	Testsubtraktion
		0001			Subtrahiere
			1111		MQR + 1
2					Zählen
	0	0011	1110		Schieben
				pos	Testsubtraktion
		0000			Subtrahieren
			1111		MQR + 1
1					Zählen
	0	0001	1110		Schieben
				neg	Testsubtraktion
0					Zählen
		0001 Rest	1110 Quotient		Ergebnisse

Bei der Umwandlung von Dualzahlen in BCD-codierte Dezimalzahlen muß die Dualzahl nacheinander durch Stellenwerte zur Basis 10 dividiert werden. Bei fünfstelligen Dezimalzahlen sind dies die Divisoren 10000, 1000, 100,10 und 1.

Beispiel: Umwandlung einer 16 bit langen positiven Dualzahl

```
 Zahl : 10000 = Zehntausenderstelle
12345 :  1000 = 1                      Rest 2345

 Rest :  1000 = Tausenderstelle
2345  :  1000 = 2                      Rest  345

 Rest :   100 = Hunderterstelle
 345  :   100 = 3                      Rest   45

 Rest :    10 = Zehnerstelle
  45  :    10 = 4                      Rest    5

            5 = Einerstelle
```

Das einfachste Verfahren subtrahiert die Konstante von der umzuwandelnden Dualzahl bzw. dem Rest in einer Schleife und zählt die Durchläufe, die dann gleich der Dezimalziffer sind.

Die praktische Ausführung der in diesem Abschnitt gezeigten Verfahren richtet sich nach dem Registeraufbau des betreffenden Prozessors. Die folgenden Beispiele wurden z.T. als Unterprogramme geschrieben und setzen die Kenntnis des Abschnitts 7 voraus.

6.5.1 Beispiele für den KIM-1 **KIM**

a.) Multiplikation von zwei 8 bit langen positiven Dualzahlen zu einem 16 bit langen Produkt. Das folgende Testprogramm ruft ein unterbrechungsfestes (re-entrant) Unterprogramm auf, das im X-Register den Multiplikanden und im Y-Register den Multiplikator übernimmt und das Ergebnis mit dem werthöheren Byte im X-Register und dem wertniedrigeren im Y-Register zurückgibt. Die Hilfsspeicherstellen liegen im Stapel, der Akkumulator und das Bedingungsregister werden gerettet und bleiben daher unverändert.

Testwerte: $10 X $20 = $0200

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			*=$0000	
0000	10	KAND	.BYTE	S10
0001	20	ATOR	.BYTE	S20
0002		PROH	*=*+1	
0003		PROL	*=*+1	
			*=$0300	
0300	AE	START	LDX	KAND
	00			
	00			
0303	AC		LDY	ATOR
	01			
	00			
0306	20		JSR	MULT
	20			
	03			
0309	8E		STX	PROH
	02			
	00			
030C	8C		STY	PROL
	03			
	00			
030F	4C	OTTO	JMP	OTTO
	0F			
	03			

Unterprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			*=$0320	
0320	08	MULT	PHP	
0321	48		PHA	
0322	D8		CLD	
0323	8A		TXA	
0324	48		PHA	
0325	98		TYA	
0326	48		PHA	
0327	A9		LDA	#$00
	00			
0329	48		PHA	
032A	48		PHA	
032B	BA		TSX	
032C	AO		LDY	#$08
	08			
032E	1E	UTE	ASL	$0101,X
	01			
	01			
0331	3E		ROL	$0102,X
	02			
	01			
0334	1E		ASL	$0103,X
	03			
	01			
0337	90		BCC	ROLF
	0F			
0339	18		CLC	
033A	BD		LDA	$0101,X
	01			
	01			
033D	7D		ADC	$0104,X
	04			
	01			
0340	9D		STA	$0101,X
	01			
	01			
0343	90		BCC	ROLF
	03			
0345	FE		INC	$0102,X
	02			
	01			
0348	88	ROLF	DEY	
0349	DO		BNE	UTE
	E3			
034B	68		PLA	
034C	A8		TAY	
034D	68		PLA	
034E	AA		TAX	
034F	68		PLA	
0350	68		PLA	
0351	68		PLA	
0352	28		PLP	
0353	60		RTS	
			.END	

b.) Division einer 16 bit langen positiven Dualzahl durch eine 8 bit lange positive Dualzahl zu einem 8 bit langen Quotienten und einem 8 bit langen Rest.
Das folgende Testprogramm ruft ein unterbrechungsfestes (reentrant) Unterprogramm auf, das im X-und Y-Register den 16stelligen Dividenden und im Akkumulator den achtstelligen Divisor übernimmt. Ist die Division ausführbar, so enthält das X-Register den Quotienten und das Y-Register den Rest. Das Carry-Bit wird gelöscht, der restliche Inhalt des Bedingungsregisters und der Akkumulator bleiben unverändert.

Ist die Division nicht ausführbar, weil der Divisor Null oder der Quotient größer als acht bit ist, so wird das Carry-Bit gesetzt, die restlichen Register bleiben unverändert. Alle Hilfsspeicherstellen liegen im Stapel.

Testwerte: $0205 : $10 = $20 Rest $05
Im Fehlerfall setzt das Hauptprogramm die Speicherstelle FEHL auf FF.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			*=$0000	
0000	02	DIVH	.BYTE	$02
0001	05	DIVL	.BYTE	$05
0002	10	DSOR	.BYTE	$10
0003		QUOT	*=*+1	
0004		REST	*=*+1	
0005	00	FEHL	.BYTE	$00
			*=$0300	
0300	AE	START	LDX	DIVH
	00			
	00			
0303	AC		LDY	DIVL
	01			
	00			
0306	AD		LDA	DSOR
	02			
	00			
0309	20		JSR	DIV
	20			
	03			
030C	8E		STX	QUOT
	03			
	00			
030F	8C		STY	REST
	04			
	00			
0312	90		BCC	OTTO
	05			
0314	A9		LDA	#$FF
	FF			
0316	8D		STA	FEHL
	05			
	00			
0319	4C	OTTO	JMP	OTTO
	19			
	03			

Unterprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			*=$0320	
0320	08	DIV	PHP	
0321	D8		CLD	
0322	48		PHA	
0323	8A		TXA	
0324	48		PHA	
0325	98		TYA	
0326	48		PHA	
0327	BA		TSX	
0328	BD		LDA	$0102,X
	02			
	01			
032B	DD		CMP	$0103,X
	03			
	01			
032E	90		BCC	GEHT
	07			
0330	68	FEHL	PLA	
0331	68		PLA	
0332	AA		TAX	
0333	68		PLA	
0334	28		PLP	
0335	38		SEC	
0336	60		RTS	
0337	A0	GEHT	LDY	#$08
	08			
0339	1E	DOLLI	ASL	$0101,X
	01			
	01			
033C	3E		ROL	$0102,X
	02			
	01			
033F	B0		BCS	HUGO
	0B			
0341	BD		LDA	$0102,X
	02			
	01			
0344	DD		CMP	$0103
	03			
	01			
0347	90		BCC	MOPS
	0D			
0349	4C		JMP	EVA
	4D			
	03			
034C	18	HUGO	CLC	
034D	FD	EVA	SBC	$0103,X
	03			
	01			
0350	9D		STA	$0102,X
	02			
	01			
0353	FE		INC	$0101,X
	01			
	01			
0356	88	MOPS	DEY	
0357	D0		BNE	DOLLI
	E0			
0359	68		PLA	
035A	AA		TAX	
035B	68		PLA	
035C	A8		TAY	
035D	68		PLA	
035E	28		PLP	
035F	18		CLC	
0360	60		RTS	
			.END	

6.5.2 Beispiele für den 6800

a.) Multiplikation von zwei acht bit langen positiven Dualzahlen zu einem 16 bit langen Produkt. Das folgende Testprogramm ruft ein unterbrechungsfestes (re-entrant) Unterprogramm auf, das im Akkumulator A den Multiplikanden und im Akkumulator B den Multiplikator übernimmt und das Produkt in beiden Registern wieder zurückgibt. Das Indexregister und Bedingungsregister werden zerstört. Alle Hilfsspeicherstellen liegen im Stapel.

Testwerte: $10 X $20 = $0200

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0000
0000	10	KAND	FCB	$10
0001	20	ATOR	FCB	$20
0002		PROH	RMB	1
0003		PROL	RMB	1
			ORG	$0180
0180	B6	START	LDA A	KAND
	00			
	00			
0183	F6		LDA B	ATOR
	00			
	01			
0186	BD		JSR	MULT
	01			
	A0			
0189	B7		STA A	PROH
	00			
	02			
018C	F7		STA B	PROL
	00			
	03			
018F	20	OTTO	BRA	OTTO
	FE			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$01A0
01A0	36	MULT	PSH A	
01A1	86		LDA A	#$08
	08			
01A3	36		PSH A	
01A4	4F		CLR A	
01A5	30		TSX	
01A6	54		LSR B	
01A7	24	MARY	BCC	IMMA
	02			
01A9	AB		ADD A	01,X
	01			
01AB	46	IMMA	ROR A	
01AC	56		ROR B	
01AD	6A		DEC	00,X
	00			
01AF	26		BNE	MARY
	F6			
01B1	31		INS	
01B2	31		INS	
01B3	39		RTS	
			END	

b.) Division einer 16 bit langen positiven Dualzahl durch eine acht bit lange positive Dualzahl zu einem acht bit langen Quotienten und einem acht bit langen Rest. Das folgende Testprogramm ruft ein unterbrechungsfestes (re-entrant) Unterprogramm auf. Der 16-stellige Dividend steht in den Akkumulatoren A und B, der achtstellige Divisor im Stapel. Das Hauptprogramm muß nach dem Aufruf des Unterprogramms die Stapeloperation wieder rückgängig machen.

Ist die Division ausführbar, so enthält der Akkumulator A den Rest und der Akkumulator B den Quotienten. Das Carry-Bit wird gelöscht. Das Indexregister wird zerstört.
Ist die Division nicht ausführbar, weil der Divisor Null oder der Quotient größer als acht bit ist, so wird das Carry-Bit gesetzt. Alle Hilfsspeicherstellen liegen im Stapel.

Testwerte: $0205 : $10 = $20 Rest $05
Im Fehlerfall setzt das Hauptprogramm die Speicherstelle FEHL auf FF.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0000
0000	02	DIVH	FCB	$02
0001	05	DIVL	FCB	$05
0002	10	ISOR	FCB	$10
0003		QUOT	RMB	1
0004		REST	RMB	1
0005		FEHL	RMB	1
			ORG	$0180
0180	96	START	LDAZ A	ISOR
	02			
0182	36		PSH A	
0183	96		LDAZ A	DIVH
	00			
0185	D6		LDAZ B	DIVL
	01			
0187	BD		JSR	DIV
	01			
	A0			
018A	97		STAZ A	REST
	04			
018C	D7		STAZ B	QUOT
	03			
018E	31		INS	
018F	86		LDA A	#$00
	00			
0191	24		BCC	MONA
	01			
0193	43		COM A	
0194	97		STAZ A	FEHL
	05			
0196	20	OTTO	BRA	OTTO
	FE			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$01A0
01A0	30	DIV	TSX	
01A1	A1		CMP A	02,X
	02			
01A3	25		BCS	ANNI
	02			
01A5	0D		SEC	
01A6	39		RTS	
01A7	36	ANNI	PSH A	
01A8	86		LDA A	#$08
	08			
01AA	36		PSH A	
01AB	30		TSX	
01AC	A6		LDA A	01,X
	01			
01AE	58	UTE	ASL B	
01AF	49		ROL A	
01B0	25		BCS	SAND
	04			
01B2	A1		CMP A	04,X
	04			
01B4	25		BCS	ROST
	04			
01B6	0C	SAND	CLC	
01B7	A2		SBC A	04,X
	04			
01B9	5C		INC	B
01BA	6A	ROST	DEC	00,X
	00			
01BC	26		BNE	UTE
	F0			
01BE	31		INS	
01BF	31		INS	
01C0	0C		CLC	
01C1	39		RTS	
			END	

6.5.3 Beispiele für den 8080/8085

8080

a.) Multiplikation von zwei acht bit langen Dualzahlen zu einem 16 bit langen Produkt. Das folgende Testprogramm ruft ein unterbrechungsfestes (re-entrant) Unterprogramm auf, das im H-Register den Multiplikator und im L-Register den Multiplikanden übernimmt und das Produkt in beiden Registern wieder zurückgibt. Alle als Hilfsspeicher benutzten Register werden gerettet.

Testwerte: 10H X 20H = 0200H

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000	10	FAK1	DB	10H
2001	20	FAK2	DB	20H
2002		PROD	DS	2
			ORG	2880H
2880	2A	START	LHLD	FAK1
	00			
	20			
2883	CD		CALL	MULT
	90			
	28			
2886	22		SHLD	PROD
	02			
	20			
2889	C3	OTTO	JMP	OTTO
	89			
	28			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2890H
2890	F5	MULT	PUSH	PSW
2891	D5		PUSH	D
2892	1E		MVI	E,08H
	08			
2894	54		MOV	D,H
2895	26		MVI	H,00H
	00			
2897	7D		MOV	A,L
2898	1F		RAR	
2899	6F		MOV	L,A
289A	7C		MOV	A,H
289B	D2	KURT	JNC	ELSE
	9F			
	28			
289E	82		ADD	D
289F	1F	ELSE	RAR	
28A0	67		MOV	H,A
28A1	7D		MOV	A,L
28A2	1F		RAR	
28A3	6F		MOV	L,A
28A4	7C		MOV	A,H
28A5	1D		DCR	E
28A6	C2		JNZ	KURT
	9B			
	28			
28A9	D1		POP	D
28AA	F1		POP	PSW
28AB	C9		RET	
			END	

Das folgende Testprogramm benutzt ein Unterprogramm, das mit den 16-Bit-Befehlen des H,L-Registerpaares arbeitet (Linksschieben und Addieren) und daher einen 16-Bit-Multiplikanden im H,L-Register mit einem 8-Bit-Multiplikator im Akkumulator zu einem 16-Bit-Ergebnis multipliziert.

Testprogramm: 0200H X 20H = 4000H

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2000H
2000	00	FAK1	DW	0200H
	02			
2002	20	FAK2	DB	20H
2003		PROD	DS	2
			ORG	2880H
2880	2A	START	LHLD	FAK1
	00			
	20			
2883	3A		LDA	FAK2
	02			
	20			
2886	CD		CALL	MULT
	90			
	28			
2889	22		SHLD	PROD
	03			
	20			
288C	C3	OTTO	JMP	OTTO
	8C			
	28			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2890H
2890	C5	MULT	PUSH	B
2891	D5		PUSH	D
2892	F5		PUSH	PSW
2893	EB		XCHG	
2894	21		LXI	H,0000H
	00			
	00			
2897	06		MVI	B,08H
	08			
2899	29	BABA	DAD	H
289A	07		RLC	
289B	D2		JNC	CATO
	9F			
	28			
289E	19		DAD	D
289F	05	CATO	DCR	B
28A0	C2		JNZ	BABA
	99			
	28			
28A3	F1		POP	PSW
28A4	D1		POP	D
28A5	C1		POP	B
28A6	C9		RET	
			END	

b) Division einer 16 bit langen positiven Dualzahl durch eine acht bit lange positive Dualzahl zu einem acht bit langen Quotienten und einem acht bit langen Rest. Das folgende Testprogramm ruft ein unterbrechungsfestes (re-entrant) Unterprogramm auf. Der 16-stellige Dividend steht im Registerpaar H,L, der achtstellige Divisor im Akkumulator.
Ist die Division ausführbar, so enthält das H-Register den Rest und das L-Register den Quotienten. Das Carry-Bit wird gelöscht.

Ist die Division nicht ausführbar, weil der Divisor Null oder der Quotient größer als acht bit ist, so wird das Carry-Bit gesetzt.

Testwerte: 0205H : 10H = 20H Rest 05H.
Im Fehlerfall setzt das Hauptprogramm die Speicherstelle FEHL auf FF.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2000H
2000	05	DIVD	DW	0205H
	02			
2002	10	ISOR	DB	10H
2003		QUOT	DS	1
2004		REST	DS	1
2005		FEHL	DS	1
			ORG	2880H
2880	2A	START	LHLD	DIVD
	00			
	20			
2883	3A		LDA	ISOR
	02			
	20			
2886	CD		CALL	DIV
	A0			
	28			
2889	22		SHLD	QUOT
	03			
	20			
288C	3E		MVI	A,00H
	00			
288E	D2		JNC	EDAM
	92			
	28			
2891	3D		DCR	A
2892	32	EDAM	STA	FEHL
	05			
	20			
2895	C3	PUPS	JMP	PUPS
	95			
	28			

Unterprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	28A0H
28A0	C5	DIV	PUSH	B
28A1	47		MOV	B,A
28A2	0E		MVI	C,08H
	08			
28A4	7C		MOV	A,H
28A5	B8		CMP	B
28A6	DA		JC	ANNI
	AC			
	28			
28A9	C1		POP	B
28AA	37		STC	
28AB	C9		RET	
28AC	29	ANNI	DAD	H
28AD	DA		JC	SAND
	B5			
	28			
28B0	7C		MOV	A,H
28B1	B8		CMP	B
28B2	DA		JC	ROST
	BA			
	28			
28B5	7C	SAND	MOV	A,H
28B6	B7		ORA	A
28B7	98		SBB	B
28B8	67		MOV	H,A
28B9	23		INX	H
28BA	0D	ROST	DCR	C
28BB	C2		JNZ	ANNI
	AC			
	28			
28BE	C1		POP	B
28BF	B7		ORA	A
28C0	C9		RET	
			END	

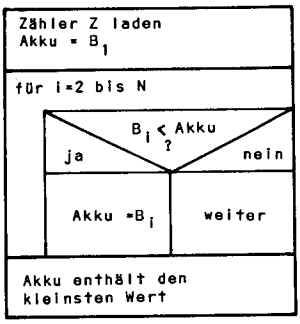
6.6 Die Arbeit mit Tabellen A

Bisher wurde gezeigt, wie sich eine Ausgangsgröße aus einer Eingangsgröße berechnen läßt. Dazu dienten logische und arithmetische Funktionen sowie Schiebebefehle. Bei vielen Anwendungen, die Meßwerte erfassen und auswerten, müssen Daten in Tabellen verarbeitet werden, weil es keinen rechnerischen Zusammenhang zwischen der Eingangs- und der Ausgangsgröße gibt. Es folgen einige häufig verwendete Verfahren der Tabellenverarbeitung.

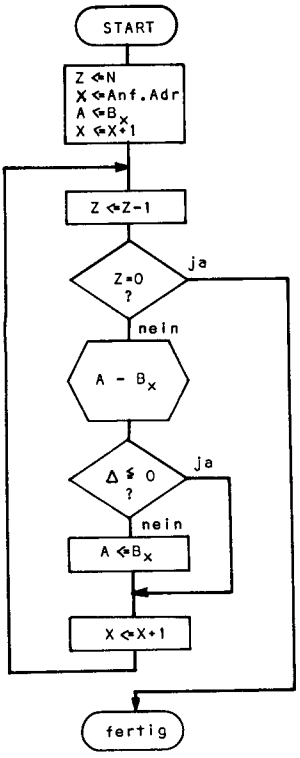
a. Suchen des kleinsten (bzw. größten)Wertes

Gegeben sei ein Bereich B bestehend aus N Werten, der nach dem kleinsten Wert durchsucht werden soll. Benötigt werden ein Zähler Z für die Kontrolle der Schleifendurchläufe, ein Indexregister X für die Adreßrechnung und ein Akkumulator A, der den jeweils kleinsten Wert aufnimmt. Der erste Wert wird willkürlich zum kleinsten erklärt und mit allen nachfolgenden verglichen. Gibt es einen kleineren, so wird dieser in den Akkumulator gebracht. Am Ende der Schleife enthält der Akkumulator den kleinsten Wert. Entsprechendes gilt, wenn der größte Wert zu suchen ist.

Programmblockplan



Programmablaufplan

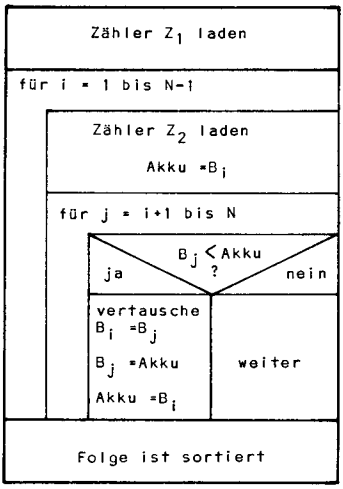


b. Sortieren nach der Größe (aufsteigend bzw. absteigend)

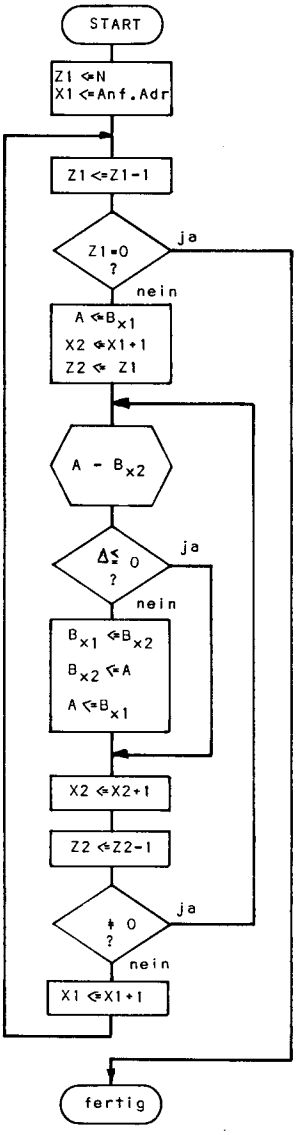
Gegeben sei ein Bereich B aus N Werten, der aufsteigend zu sortieren ist; der kleinste Wert soll also im ersten Speicherplatz und der größte im letzten liegen. Das Verfahren arbeitet mit zwei ineinander geschachtelten Schleifen und benötigt zwei Indexregister X_1 und X_2, zwei Zähler Z_1 und Z_2 sowie einen Akkumulator, der ähnlich wie im Abschnitt a.) den jeweils kleinsten Wert enthält. Die äußere Schleife (X_1 und Z_1) erklärt den jeweils obersten Wert zum kleinsten und durchsucht alle nachfolgenden in der inneren Schleife (X_2 und Z_2) nach einem noch klei-

neren. Gibt es einen, so werden der durch die äußere Schleife adressierte oben liegende Wert und der durch die innere Schleife adressierte kleinere Wert vertauscht. Die äußere Schleife läuft von I = 1 bis N-1, die innere von J = I+1 bis N. Nach dem Ablauf beider Schleifen ist der Bereich sortiert.

Programmblockplan



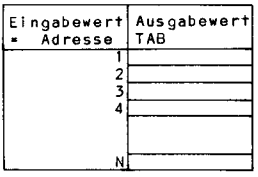
Programmablaufplan



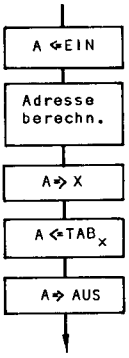
c. Direkter Tabellenzugriff

Gegeben sei eine Funktion A = f (E), bei der der Eingabewert E ganze positive Zahlen von 1 bis N, der Ausgabewert A beliebige Werte von ein byte Länge annehmen kann. Unter dieser Voraussetzung kann man den Eingabewert E als Adresse in einer Tabelle ansehen, die nur die Ausgabewerte enthält. Der Eingabewert 5 würde also dem 5. Tabellenwert entsprechen.

Tabelle



Programmablaufplan

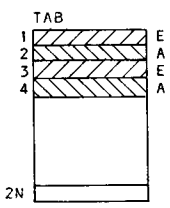


Sind die Eingabewerte nicht gleich der Adresse oder sind die Ausgabewerte länger als ein byte, so sind Adreßrechnungen erforderlich.

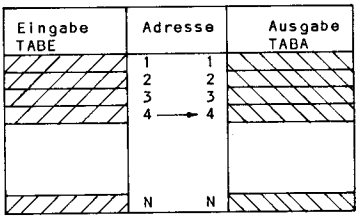
d. Lineares Tabellensuchen

Gegeben sei eine Funktion A = f (E), bei der der Eingabewert E nicht als Adresse des Ausgabewertes A dienen kann. Dann müssen auch die Eingabewerte in einer Tabelle untergebracht werden. Eine gemischte Tabelle ist mit der Schrittweite 2 nach dem Eingabewert zu durchsuchen. Arbeitet man mit zwei Tabellen, so genügt die Schrittweite 1, die Adresse des gefundenen Wertes zeigt auf den Ausgabewert.

Gemischte Tabelle

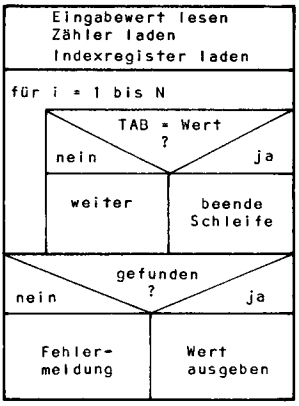


getrennte Tabellen

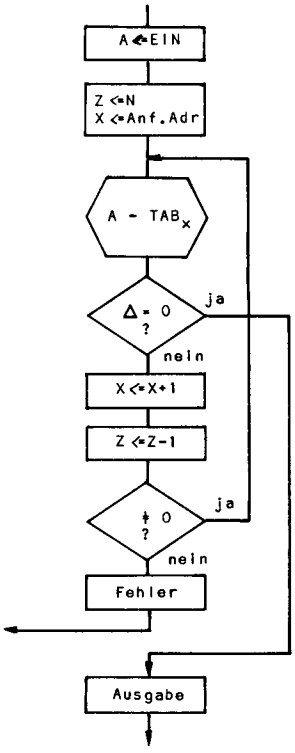


Das folgende Beispiel zeigt, wie man eine Tabelle TAB bestehend aus N Elementen durchsucht.

Programmblockplan



Programmablaufplan



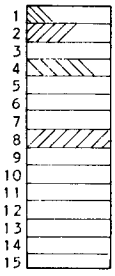
Bei einer gleichmäßigen Verteilung der Listenelemente sind bei jedem Durchlauf N/2 Speicherzugriffe erforderlich. Daher ist es günstig, häufig auftretende Werte an den Anfang zu legen.

e. Binäres Tabellensuchen

Sind die Elemente einer zu durchsuchenden Tabelle aufsteigend der Größe nach sortiert, so kann man in der Mitte beginnen und sofort feststellen, ob sich der gesuchte Wert im oberen oder unteren Teil befindet. Durch weitere Halbierungen läßt sich der Wert wesentlich schneller finden, als wenn man die Tabelle linear durchsucht. Das Verfahren wird besonders einfach, wenn man eine Tabelle hat, die bei jeder Teilung zwei gleiche Hälften ergibt. Dies trifft zu für N = 3,7,15,31,63,127,255 usw.; allgemein $2^K - 1$ für k = 2,3,4, usw.

Beispiel

Gegeben sei eine Tabelle aus 15 Elementen



Vergleicht man den gesuchten Wert mit dem in der Mitte liegenden 8. Element, so gibt es drei Möglichkeiten:

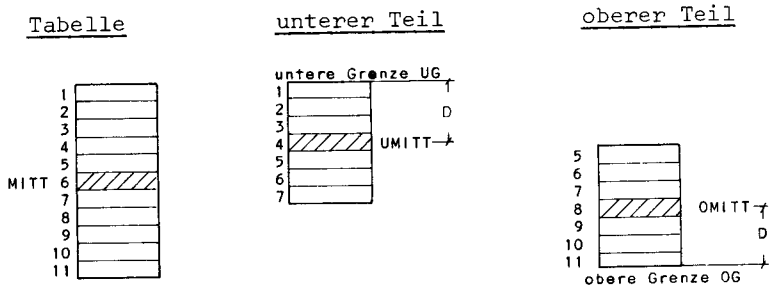
a.) Beide Werte sind gleich: Der Wert ist gefunden.
b.) Der gesuchte Wert ist größer: im oberen Bereich weitersuchen.
c.) Der gesuchte Wert ist kleiner: im unteren Bereich weitersuchen.

Der untere Bereich besteht aus 7 Elementen, die durch das 4. Element in zwei Dreiergruppen unterteilt werden. Das 2. Element ergibt zwei Einergruppen. Nach maximal vier Speicherzugriffen (im Mittel zwei) ist der Wert gefunden, wenn er sich überhaupt in der Tabelle befindet.

Will man das Verfahren auf beliebig große Tabellen übertragen, so muß man diese so unterteilen, daß zwei Bereiche entstehen, die der Bedingung N = 3,7,15 usw. genügen. Dabei dürfen sich die Bereiche überlappen.

Beispiel:

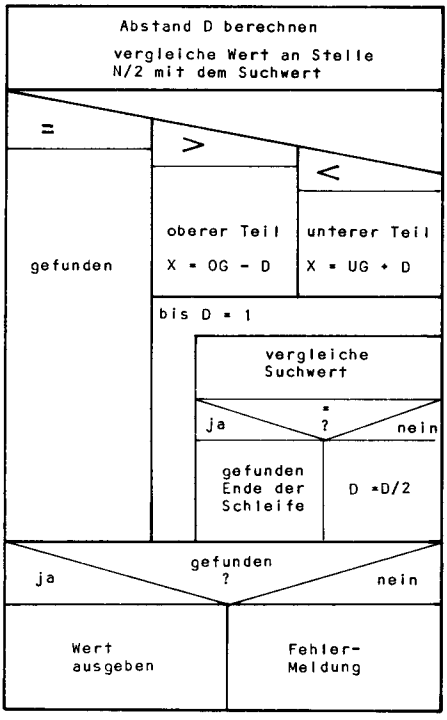
Gegeben sei eine Tabelle aus 11 Elementen. Eine erste Prüfung genau in der Mitte entscheidet, ob in der oberen Siebenergruppe oder in der unteren Siebenergruppe weiterzusuchen ist.



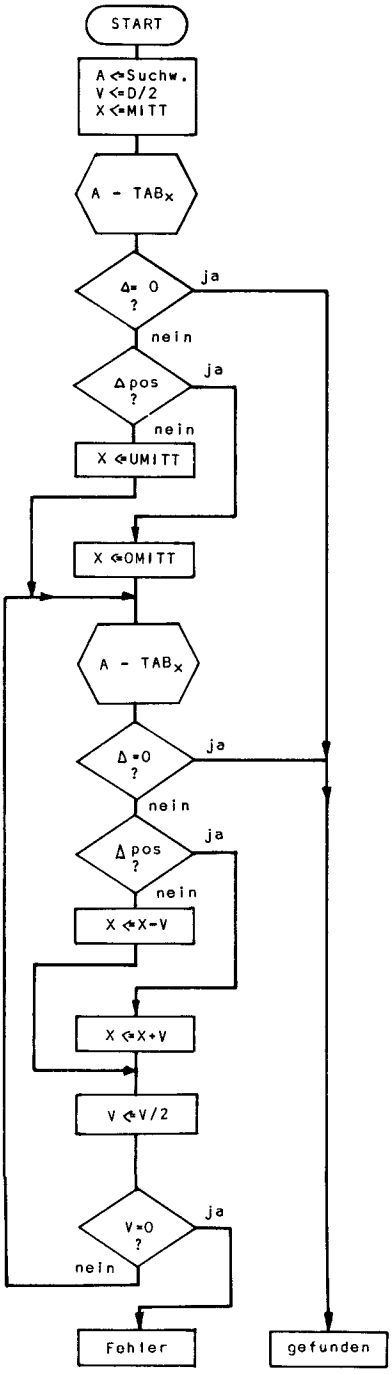
Allgemein entscheidet also die Abfrage in der Mitte, ob im oberen oder im unteren Bereich weiterzusuchen ist. Zur Berechnung der in der Mitte liegenden Adressen jedes Teils muß man feststellen, welches die nächstkleinere Zweierpotenz der Anzahl N ist. Für N = 11 ist dies die 8.

Der Abstand der Mittelelemente von der oberen bzw. unteren Grenze ist dann die Hälfte dieser Zweierpotenz, für N = 11 also 8/2 = 4.

Programmblockplan



Programmablaufplan



Übungen zum Abschnitt 6.

1.Aufgabe

Von den Kippschaltern einer Schaltung nach Bild 48 ist ein beliebiges Bitmuster einzulesen und verändert wieder auszugeben.

Bit 0 und 1 sind zu komplementieren.
Bit 2 und 3 bleiben unverändert.
Bit 4 und 5 sind konstant 0 zu setzen.
Bit 6 und 7 sind konstant 1 zu setzen.

2.Aufgabe

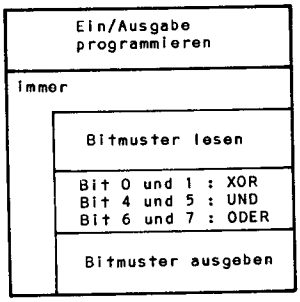
Über die vier werthöheren Schalter einer Schaltung nach Bild 48 ist der Multiplikator, über die vier wertniedrigeren Schalter der Multiplikand als positive Dualzahl einzugeben. An den Leuchtdioden ist das Produkt anzuzeigen.

3.Aufgabe

An den Kippschaltern einer Schaltung nach Bild 48 ist eine positive Dualzahl kleiner 100 einzugeben, die als zweistellige BCD-Zahl auf den Leuchtdioden anzuzeigen ist.

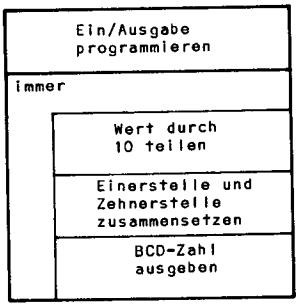
Programmblockpläne

1.Aufgabe



2.Aufgabe

3.Aufgabe



7. Unterprogramme und Programmunterbrechungen

A

Die bisher behandelten Übungsprogramme waren kurz und einfach zu überschauen. Sie behandelten immer nur ein Teilproblem. In der Praxis bestehen die Programme aus mehreren Teilaufgaben, die man oft zu Unterprogrammen zusammenfaßt. Das Hauptprogramm verbindet die Arbeit der Unterprogramme. Auf die Elektronik übertragen sind die Unterprogramme Baugruppen. Das Hauptprogramm bildet den Verdrahtungsrahmen.

Der Abschnitt 5 zeigte Programmverzweigungen, bei denen das Programm laufend die Geräte - in den Beispielen einfache Schalter - abfragte. Bei schneller Datenübertragung ist es zeitlich vorteilhafter, wenn das Gerät seine Eingabewünsche über eine Steuerleitung (Interrupt) meldet. Dabei wird das laufende Programm unterbrochen, der Mikrocomputer bedient das Gerät und kehrt dann wieder in das unterbrochene Programm zurück.

Bei Aufruf eines Unterprogramms und beim Starten eines Unterbrechungsprogramms, das einen Interrupt bedient, muß der Stand des Befehlszählers gerettet werden, um eine Rückkehr in das unterbrochene Programm zu ermöglichen. Da bei geschachtelten Unterprogrammen oder Interrupts oft mehrere Rücksprungadressen gespeichert werden müssen, legt man diese im Stapelbereich ab, der beliebig viele Rücksprungadressen nacheinander aufnehmen kann. Der Programmierer bestimmt beim Start des Programms die Anfangsadresse dieses Stapels und kann nun fortlaufend Daten und Rücksprungadressen ablegen und wieder zurückholen.

7.1 Die Stapelbefehle

Der Stapel ist ein Bereich im Schreib/Lese-Speicher (RAM) zur Aufnahme von Daten und Adressen.
Der Stapelzeiger ist ein Register des Mikroprozessors. Er enthält die Adresse des Speicherbytes im Stapel, das als nächstes gelesen oder beschrieben wird. Nach jedem Schreibvorgang wird der Stapelzeiger um 1 erniedrigt, nach jedem Lesevorgang um 1 erhöht.

Das bedeutet:

a. Mehrere aufeinanderfolgende Schreibbefehle legen die Daten hintereinander im Stapel ab.
b. Mehrere aufeinanderfolgende Lesebefehle holen hintereinanderliegende Daten aus dem Speicher.
c. Ein Lesebefehl holt die zuletzt eingeschriebenen Daten aus dem Stapel.

Bei den Befehlen "Stapel schreiben" und "Stapel lesen" steht die Adresse der Daten im Stapelzeiger, es sind daher 1-Byte-Befehle ohne Adreßteil.
Bei jedem Unterprogrammaufruf und bei jedem Interrupt wird der Inhalt des Befehlszählers von der Prozessorsteuerung in den Stapel gerettet. Die Rücksprungbefehle holen diesen Wert aus dem Stapel zurück und setzen damit das Programm an der unterbrochenen Stelle fort.

Das Betriebsprogramm, mit dem der Benutzer in den meisten Fällen sein Programm testet, wird bei jedem RESET neu gestartet. Es lädt den Stapelzeiger mit einem Anfangswert und benutzt den Stapel für die Kontrolle der Benutzerprogramme. Der Benutzer kann den Stapel des Betriebsprogramms mitverwenden oder in dem ihm zur Verfügung stehenden RAM-Bereich einen eigenen Stapel aufbauen.

7.1.1 Die Stapelbefehle des KIM-1 KIM

Das Stapel-Adreßregister, der Stapelzeiger, des Mikroprozessors 6502 ist acht bit lang und enthält das niederwertige Byte der Stapeladresse. Bei allen Stapeloperationen setzt der Prozessor das höherwertige Byte der Adresse 01; der Stapel liegt also immer in Seite 1.

Bild 106 zeigt den Aufbau des Stapels und des Stapelzeigers an einem Beispiel.

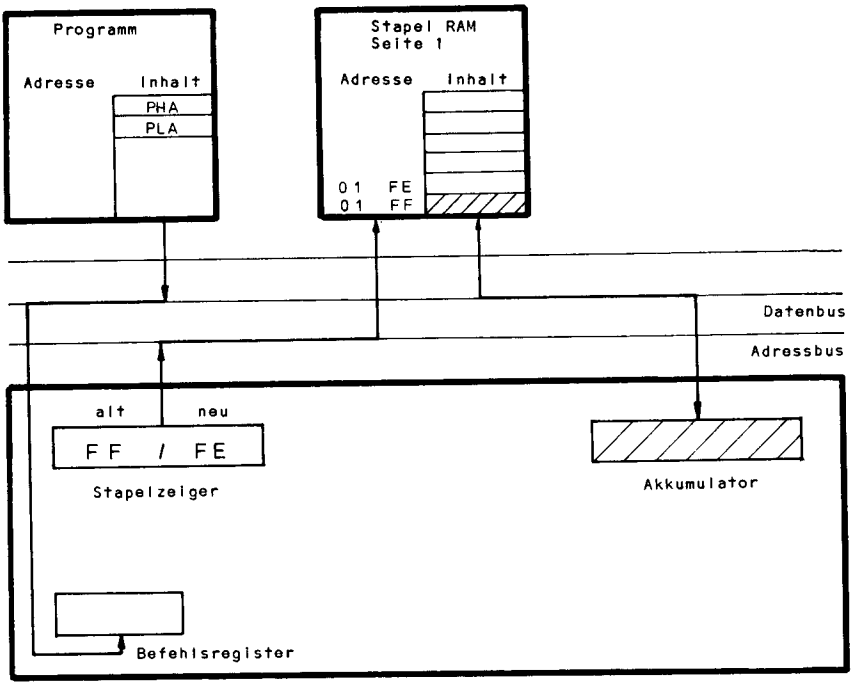


Bild 106: Stapel und Stapelzeiger des KIM-1

Der Stapelzeiger adressiert die Speicherstelle 01FF. Der Befehl PHA "Schreibe Stapel" bringt den Inhalt des Akkumulators in die Speicherstelle 01FF und vermindert den Stapelzeiger um 1, sodaß er danach auf die Speicherstelle 01FE zeigt.

Der Befehl PLA "Lesen Stapel" erhöht zuerst den Stapelzeiger um 1 auf FF und holt dann den Inhalt der damit adressierten Speicherstelle 01FF zurück in den Akkumulator. Die folgenden Befehle verbinden den Akkumulator bzw. das Bedingungsregister mit dem Stapel.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen N V B D I Z C	
PHA		Stapelzeiger	48	1	3		Akkumulator nach Stapel, Zeiger ← Zeiger - 1
PHP		Stapelzeiger	08	1	3		Bedingungs. nach Stapel, Zeiger ← Zeiger -1
PLA		Stapelzeiger	68	1	4	X X	Zeiger ← Zeiger + 1, Akkumulator vom Stapel
PLP		Stapelzeiger	28	1	4	vom Stapel	Zeiger ← Zeiger + 1, Bedingungs. vom Stapel

PH ist eine Abkürzung für PusH = in den Stapel drücken,
PL bedeutet PuLl = aus dem Stapel ziehen.

Nach dem Einschalten der Versorgungsspannung ist der Zustand der RAM-Speicher und Register des Mikroprozessors nicht definiert. Ein mit RESET gestartetes Programm muß zunächst den Stapelzeiger mit einem Anfangswert laden. Da beim Mikroprozessor 6502 der Stapel immer in Seite 1 liegt, stehen bei einem Ausgangswert FF höchstens 256 Bytes für den Stapel zur Verfügung.

Beispiel:

Das folgende Testprogramm lädt den Stapelzeiger über das X-Register mit dem Ausgangswert FF und rettet anschließend den Akkumulator und das Y-Register in den Stapel.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       *=$0300
0300    A2      START  LDX      #$FF
        FF
0302    9A             TXS
0303    48             PHA
0304    98             TYA
0305    48             PHA
0306    4C      MAUS   JMP      MAUS
        06
        03
                       .END
```

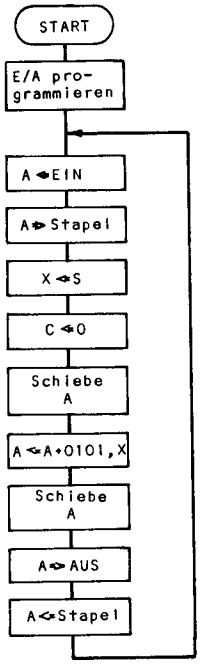
Das Betriebsprogramm des KIM-1 setzt bei jedem RESET den Stapelzeiger auf FF, der Benutzer kann mit dem damit adressierten Stapelbereich arbeiten und braucht den Stapelzeiger nicht zu laden.

Programme müssen oft "re-entrant", d.h. unterbrechungsfest geschrieben werden. Dies bedeutet, daß sie jederzeit durch einen Unterprogrammaufruf oder Interrupt unterbrochen werden können, ohne daß die gerade bearbeiteten Daten verloren gehen. Diese dürfen also nur in den Registern des Mikroprozessors oder im Stapel, nicht aber auf festen Speicheradressen liegen. Der Befehl TSX überträgt den Inhalt des Stapelzeigers in das X-Register. Damit können die im Stapel liegenden Daten auch mit indizierter Adressierung angesprochen werden. Da der Stapel immer in Seite 1 liegt, steht in den Befehlen ein konstanter Abstand, zu dem der varia-

ble Inhalt des X-Registers entsprechend dem Stand des Stapelzeigers dazuaddiert wird. Der Abstand $0101 im Befehl, indiziert mit dem X-Register, adressiert das letzte in den Stapel geschriebene Byte.

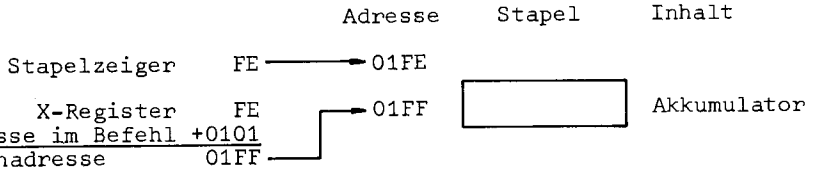
Beispiel:

Das folgende Testprogramm multipliziert eine im Akkumulator stehende positive Dualzahl kleiner 26_{10} mit 10_{10}. Dazu wird der Akkumulatorinhalt zunächst in den Stapel gerettet, um zwei bit nach links geschoben, über indizierte Adressierung mit dem alten, im Stapel liegenden Wert addiert und nochmals um ein bit nach links geschoben. Die beiden Indexregister können nicht als Zwischenspeicher dienen, da mit ihnen keine arithmetischen Operationen möglich sind.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0300	
0300	A9	START	LDA	#$FF
	FF			
0302	8D		STA	$1701
	01			
	17			
0305	A9		LDA	#$00
	00			
0307	8D		STA	$1703
	03			
	17			
030A	D8		CLD	
030B	AD	MAX	LDA	$1702
	02			
	17			
030E	48		PHA	
030F	BA		TSX	
0310	18		CLC	
0311	2A		ROLA	
0312	2A		ROLA	
0313	7D		ADC	$0101,X
	01			
	01			
0316	2A		ROLA	
0317	8D		STA	$1700
	00			
	17			
031A	68		PLA	
031B	4C		JMP	MAX
	0B			
	03			
			.END	

Bei der indizierten Adressierung ergibt sich folgende Adreßrechnung unter der Annahme, daß der Stapelzeiger vor dem Start des Programms auf FF stand:



Der Befehl PLA am Ende der Schleife bringt den Stapelzeiger wieder auf seinen alten Wert.

7.1.2 Die Stapelbefehle des 6800

Das Stapel-Adreßregister (Stapelzeiger) des Mikroprozessors 6800 ist 16 bit lang und enthält die vollständige Stapeladresse. Der Stapel kann daher beliebig in den RAM-Bereich gelegt werden. Bild 107 zeigt den Aufbau des Stapelzeigers und Stapels mit einem Beispiel.

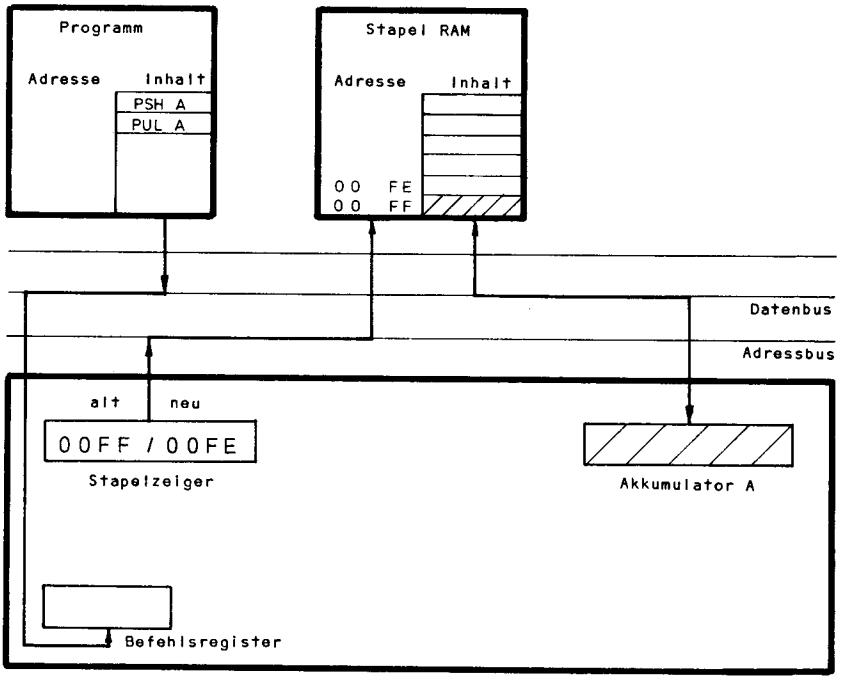


Bild 107: Stapel und Stapelzeiger des 6800

Der Stapelzeiger adressiert die Speicherstelle 00FF. Der Befehl PSH A "Schreibe Stapel" bringt den Inhalt des Akkumulators A in die Speicherstelle 00FF und vermindert dann den Stapelzeiger um 1, sodaß er danach auf die Speicherstelle 00FE zeigt.

Der Befehl PUL A "Lesen Stapel" erhöht zuerst den Stapelzeiger von 00FE auf 00FF und holt dann den Inhalt der damit adressierten Speicherstelle 00FF in den Akkumulator A.

Die folgenden Befehle verbinden die beiden Akkumulatoren mit dem Stapel.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
PSH A		Stapelzeiger	36	1	4		Akku A nach Stapel, Zeiger ← Zeiger - 1
PSH B		Stapelzeiger	37	1	4		Akku B nach Stapel, Zeiger ← Zeiger - 1
PUL A		Stapelzeiger	32	1	4		Zeiger ← Zeiger + 1, Akku A vom Stapel
PUL B		Stapelzeiger	33	1	4		Zeiger ← Zeiger + 1, Akku B vom Stapel

PSH bedeutet PuSH = in den Stapel drücken, PUL bedeutet PULl = aus dem Stapel ziehen.

Nach dem Einschalten der Versorgungsspannung ist der Zustand der RAM-Speicher und der Register des Mikroprozessors nicht definiert. Ein mit RESET gestartetes Programm muß zunächst den Stapelzeiger mit einem Anfangswert laden.

Beispiel:

Das folgende Testprogramm lädt den Stapelzeiger mit dem Ausgangswert 00FF und rettet anschließend alle Register in den Stapel. Zum Speichern des X-Registers sind zwei Hilfsspeicherstellen erforderlich.

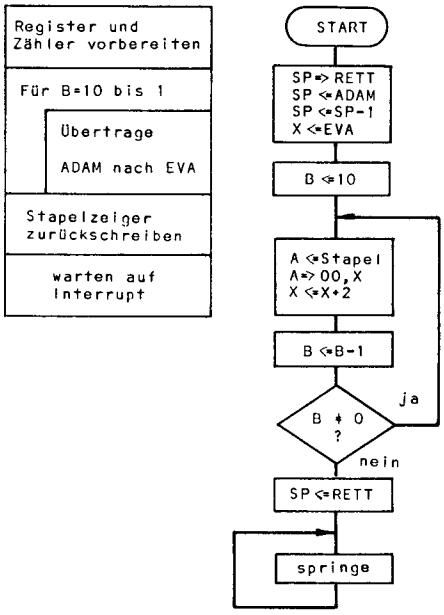
```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      $0000
0000            HILF1  RMB      1
0001            HILF2  RMB      1

                       ORG      $0180
0180      8E    START  LDS      #$00FF
          00
          FF
0183      36           PSH A
0184      37           PSH B
0185      EF           STX      HILF1
          00
          00
0188      B6           LDA A    HILF1
          00
          00
018B      F6           LDA B    HILF2
          00
          01
018E      36           PSH A
          37           PSH B
0190      20    OTTO   BRA      OTTO
          FE
                       END
```

Der Mikroprozessor 6800 hat nur ein Indexregister für die Berechnung von Datenadressen. Deshalb mußte das Indexregister bei der Übertragung von zwei Bereichen dauernd zwischengespeichert werden. Das folgende Beispiel zeigt, wie man einen Bereich mit dem Indexregister und den anderen mit dem Stapelzeiger adressiert.

Beispiel:

Man übertrage zehn hintereinanderliegende Bytes aus einem Bereich ADAM (Anfangsadresse 0000) in jedes zweite Byte des Bereiches EVA (Anfangsadresse 0100).

Adressiert man den Herkunftsbereich mit dem Stapelzeiger, so beginnt man mit der Anfangsadresse, da der Stapelzeiger beim Lesen aufwärts zählt. Adressiert man den Zielbereich mit dem Stapelzeiger, so muß man mit der Endadresse beginnen, da der Stapelzeiger beim Schreiben abwärts zählt.



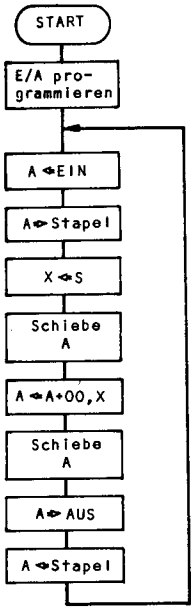
Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0000
0000		ADAM	RMB	10
000A		RETT	RMB	2
			ORG	$0100
0100		EVA	RMB	20
			ORG	$0180
0180	BF	START	STS	RETT
	00			
	0A			
0183	8E		LDS	#ADAM
	00			
	00			
0186	34		DES	
0187	CE		LDX	#EVA
	01			
	00			
018A	C6		LDA B	#$0A
	0A			
018C	32	OTTO	PUL A	
018D	A7		STA A	00,X
	00			
018F	08		INX	
0190	08		INX	
0191	5A		DEC B	
0192	26		BNE	OTTO
	F8			
0194	BE		LDS	RETT
	00			
	0A			
0197	20	SUSI	BRA	SUSI
	FE			
			END	

In dem Beispiel adressiert der Stapelzeiger den Herkunftsbereich ADAM, das Indexregister mit der Schrittweite 2 den Zielbereich EVA. Der Akkumulator B zählt die Schleifendurchläufe. Der alte Stand des Stapelzeigers wird am Anfang gerettet und am Ende zurückgeschrieben.

Programme, die unterbrechungsfest (re-entrant) geschrieben sind, können jederzeit durch ein Unterprogramm oder einen Interrupt unterbrochen werden, ohne daß die gerade bearbeiteten Daten verloren gehen. Diese dürfen also nicht in festen Speicheradressen liegen, sondern müssen in den Registern des Mikroprozessors oder im Stapel abgelegt werden. Der Befehl TSX überträgt den um 1 erhöhten Inhalt des Stapelzeigers in das X-Register, das damit genau auf das letzte in den Stapel geschriebene Byte zeigt. Damit können alle im Stapel liegenden Werte über das Indexregister adressiert werden.

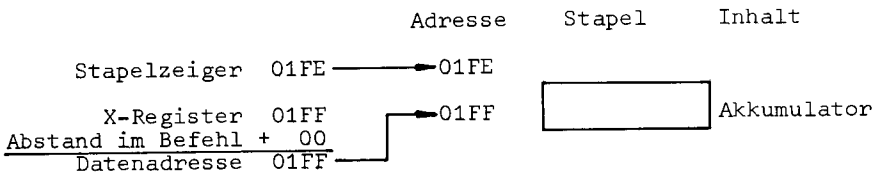
Beispiel:

Das folgende Testprogramm multipliziert eine im Akkumulator A stehende positive Dualzahl kleiner 26_{10} mit 10_{10}. Dazu wird der Akkumulator zunächst gerettet, um zwei bit nach links geschoben und dann mit dem geretteten Wert addiert. Die Summe ist nochmals um 1 bit nach links zu schieben. Obwohl man bequemer den Akkumulator A in den Akkumulator B rettet, benutzt das Beispiel den Stapel, um die Adressierung des Stapels über das Indexregister zu zeigen.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0180
0180	73	START	COM	$8004
	80			
	04			
0183	86		LDA A	#$04
	04			
0185	B7		STA A	$8005
	80			
	05			
0188	B7		STA A	$8007
	80			
	07			
018B	B6	EVA	LDA A	$8006
	80			
	06			
018E	36		PSH A	
018F	30		TSX	
0190	48		ASL A	
0191	48		ASL A	
0192	AB		ADD A	00,X
	00			
0194	48		ASL A	
0195	B7		STA A	$8004
	80			
	04			
0198	32		PUL A	
0199	7E		JMP	EVA
	01			
	8B			
			END	

Bei der indizierten Adressierung ergibt sich folgende Adreßrechnung unter der Annahme, daß der Stapelzeiger vor dem Start des Programms auf der Adresse 01FF stand:



Der Befehl PUL A am Ende der Schleife bringt den Stapelzeiger wieder auf seinen alten Wert.

7.1.3 Die Stapelbefehle des 8080/8085 **8080**

Das Stapel-Adreßregister (Stapelzeiger) der Mikroprozessoren 8080 und 8085 ist 16 bit lang und enthält die vollständige Stapeladresse. Der Stapel kann daher beliebig im RAM-Bereich liegen. Bild 108 zeigt den Aufbau des Stapels und des Stapelzeigers mit einem Beispiel.

Der Stapelzeiger adressiert die Speicherstelle 2080. Die Stapelbefehle transportieren immer ein Registerpaar. Das Bedingungsregister (PSW) und der Akkumulator bilden dabei ebenfalls ein Paar.

Der Befehl PUSH "Schreibe Stapel" vermindert zunächst den Stapelzeiger um 1, also vom Ausgangswert 2080 auf 207F, und speichert das erste der beiden Register unter dieser Adresse ab.

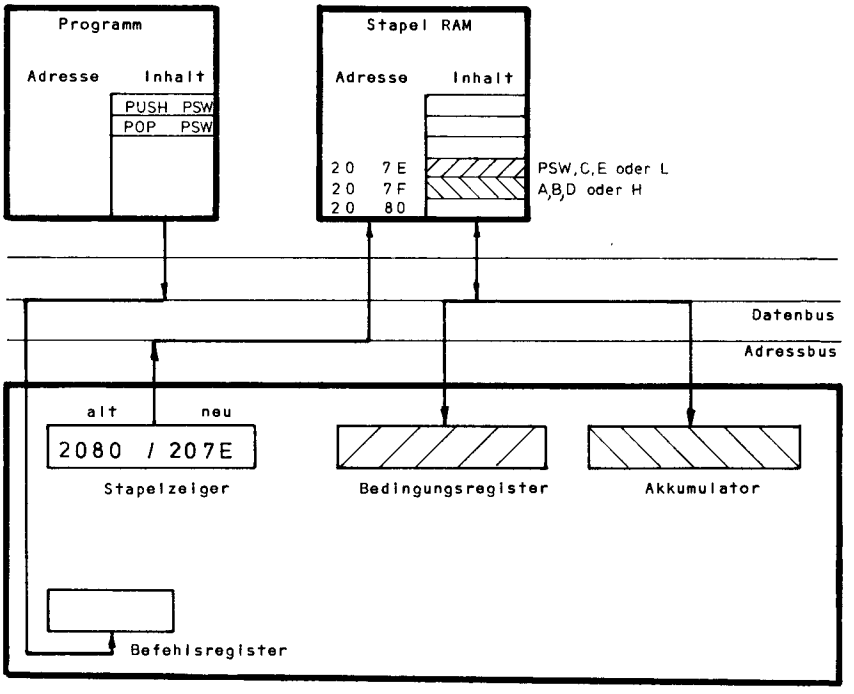


Bild 108: Stapel und Stapelzeiger des 8080/8085

Anschließend wird der Zeiger zum Speichern des zweiten Registers nochmals um 1 vermindert. Er steht dann auf der Adresse 207E.

Der Befehl POP "Lies Stapel" bringt das gerade adressierte Byte mit der Adresse 207E in das zweite Register und das folgende Byte in das erste. Der Stapelzeiger wird dabei um 2 erhöht. Er hat danach den Wert 2080.

Die folgenden Befehle greifen mit Hilfe des Stapelzeigers(SP = Stack Pointer) auf den Stapel zu. Sie transportieren ein Registerpaar. Das mit PSW adressierte Paar besteht aus dem Bedingungsregister (PSW = Processor Status Word) und dem Akkumulator.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
PUSH	B	Stapelzeiger	C5	1	10			SP←SP-1,B nach Stapel,SP←SP-1,C nach Stapel
PUSH	D	Stapelzeiger	D5	1	10			SP←SP-1,D nach Stapel,SP←SP-1,E nach Stapel
PUSH	H	Stapelzeiger	E5	1	10			SP←SP-1,H nach Stapel,SP←SP-1,L nach Stapel
PUSH	PSW	Stapelzeiger	F5	1	10			SP←SP-1,A n. Stapel,SP←SP-1,PSW nach Stapel
POP	B	Stapelzeiger	C1	1	10			C vom Stapel,SP←SP+1,B vom Stapel,SP←SP+1
POP	D	Stapelzeiger	D1	1	10			E vom Stapel,SP←SP+1,D vom Stapel,SP←SP+1
POP	H	Stapelzeiger	E1	1	10			L vom Stapel,SP←SP+1,H vom Stapel,SP←SP+1
POP	PSW	Stapelzeiger	F1	1	10	wie Stapel		PSW vom Stapel,SP←SP+1,A vom Stapel,SP←SP+1

PUSH bedeutet drücke in den Stapel, POP bedeutet hole aus dem Stapel heraus.

Der folgende Befehl XTHL vertauscht entsprechend Bild 109 den Inhalt des Registerpaares H,L mit den beiden Bytes, die zuletzt in den Stapel geschrieben wurden. Sie werden adressiert durch den Stapelzeiger und den Stapelzeiger +1.
Der Stapelzeiger wird durch den Befehl nicht verändert.

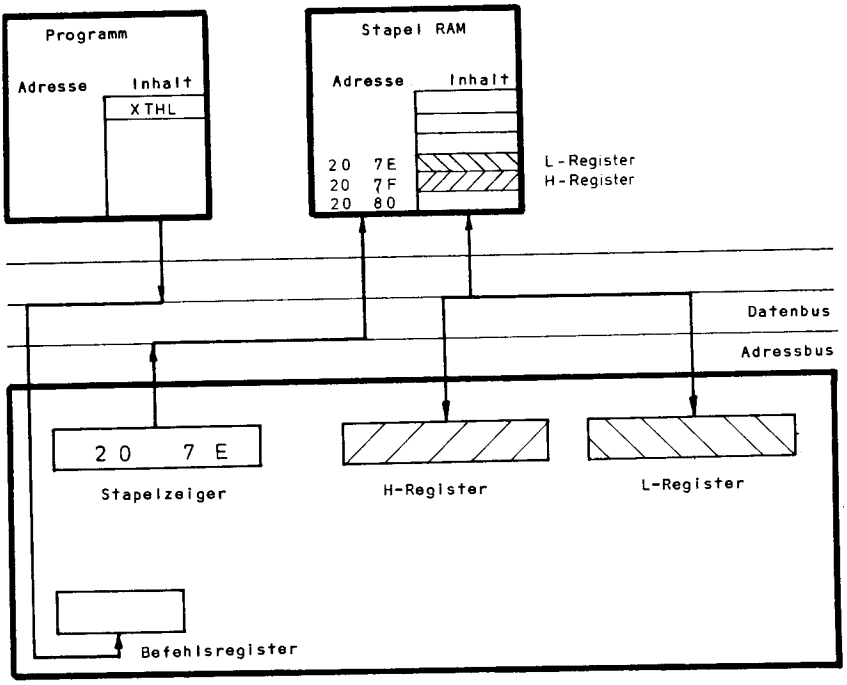


Bild 109: Der XTHL-Befehl des 8080/8085

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
XTHL		Stapelzeiger	E3	1	18			Vertausche H,L mit Stapel (Adr. SP und SP+1)

XTHL bedeutet eXchange the Top of the stack with H,L = vertausche die Spitze des Stapels mit dem Inhalt des H,L - Registers.

Beispiel:

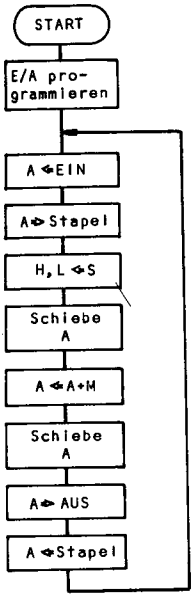
Nach dem Retten des Stapelzeigers ist ein neuer Stapelbereich mit der Ausgangsadresse 2080 aufzustellen, in den alle Register zu retten sind.

```
  Hexadezimal          Symbolisch
Adresse Inhalt  Name   Befehl   Operand

                       ORG      2000H
2000            RETT   DS       2
2002            HILF   DS       2

                       ORG      2880H
2880      22    START  SHLD     HILF
          02
          20
2883      21           LXI      H,0000
          00
          00
2886      39           DAD      SP
2887      22           SHLD     RETT
          00
          20
288A      2A           LHLD     HILF
          02
          20
288D      31           LXI      S,2080H
          80
          20
2890      C5           PUSH     B
2891      D5           PUSH     D
2892      E5           PUSH     H
2893      F5           PUSH     PSW
2894      2A           LHLD     RETT
          00
          20
2897      F9           SPHL
2898      C3    OTTO   JMP      OTTO
          98
          29
                       END
```

Unterbrechungsfest (re-entrant) geschriebene Programme können jederzeit durch ein Unterprogramm oder einen Interrupt unterbrochen werden, ohne daß die gerade bearbeiteten Daten verloren gehen. Diese müssen daher in den Registern des Mikroprozessors oder im Stapel abgelegt werden. Der Befehl DAD SP überträgt den Inhalt des Stapelzeigers in das vorher zu löschende H,L-Register, das nun genau auf das letzte in den Stapel geschriebene Byte zeigt. Über die Speicheradressierung können nun alle im Stapel liegenden Werte angesprochen werden.

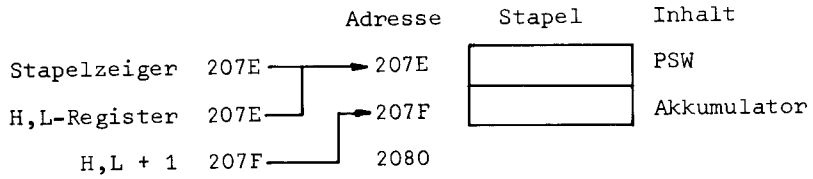
Beispiel:

Das folgende Testprogramm multipliziert eine im Akkumulator stehende positive Dualzahl kleiner 26_{10} mit 10_{10}. Dazu wird der Akkumulator zunächst gerettet, um zwei bit nach links geschoben und dann mit dem geretteten Wert addiert. Die Summe ist nochmals um ein bit nach links zu schieben. Obwohl man bequemer den Akkumulator in ein anderes Register rettet, benutzt das Beispiel den Stapel, um die Adressierung des Stapels über das Registerpaar H,L zu zeigen.



Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	D3		OUT	02H
	02			
2884	AF		XRA	A
2885	D3		OUT	03H
	03			
2887	DB	MAX	IN	01H
	01			
2889	F5		PUSH	PSW
288A	21		LXI	H,0000
	00			
	00			
288D	39		DAD	SP
288E	23		INX	H
288F	B7		ORA	A
2890	07		RLC	
2891	07		RLC	
2892	86		ADD	M
2893	07		RLC	
2894	D3		OUT	00H
	00			
2896	F1		POP	PSW
2897	C3		JMP	MAX
	87			
	28			
			END	

Unter der Annahme, daß der Stapelzeiger vor dem Start des Programms auf der Adresse 2080 stand, ergibt sich folgende Adreßverteilung:



Der Befehl POP PSW am Ende der Schleife bringt den Stapelzeiger wieder auf seinen alten Wert.

7.2 Unterprogrammtechnik

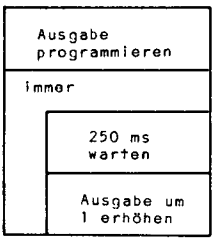
A

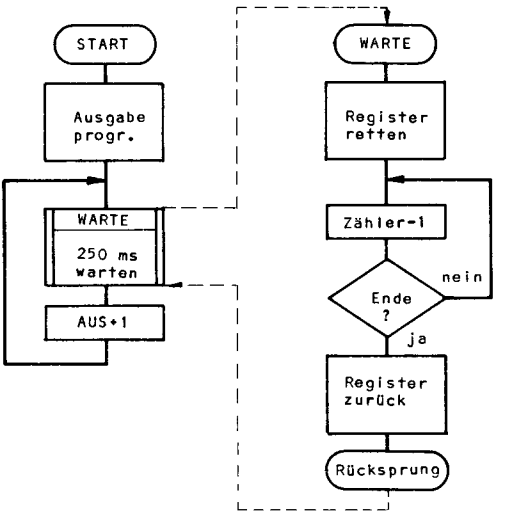
Unterprogramme sind Programmteile, die bestimmte Teilaufgaben in einem Programm lösen.

Beispiel:

Ein Unterprogramm mit dem Namen WARTE verzögert den Ablauf des Hauptprogramms um eine bestimmte Zeit.
Der Programmablaufplan stellt den Aufruf eines Unterprogramms durch ein Rechteck mit Seitenbalken dar.

Der Programmblockplan zeigt keine Einzelheiten der Programmausführung und berücksichtigt nicht, ob ein Teilproblem durch ein Unterprogramm gelöst wird oder Bestandteil des Hauptprogramms ist.





Unterprogramme bieten folgende Vorteile:

a. Das Problem wird zerlegt; das Programm wird damit übersichtlicher und läßt sich leichter testen und ändern.
b. Bereits erprobte Teillösungen können einfach einer Programmbibliothek entnommen werden.
c. Programmteile, die mehrmals im Programm benötigt werden, aber nur einmal als Unterprogramm vorhanden sind, sparen Speicherplatz und Programmierarbeit.
d. Ein Unterprogramm kann weitere Unterprogramme aufrufen. Die Schachtelung kann beliebig tief sein, da die Rücksprungadressen in den Stapel gerettet werden.

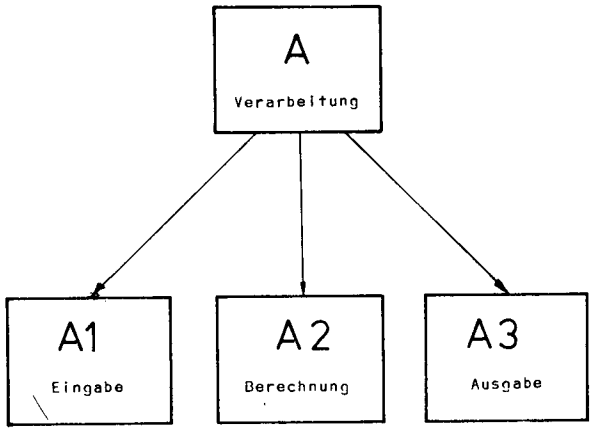
Bei der Programmierung größerer Aufgaben geht man oft nach dem "Top-down-Verfahren" vor und unterteilt dabei die Aufgabe durch schrittweise Verfeinerung in immer kleinere Teilaufgaben. Es ist nun zweckmäßig, diese Teilaufgaben als Unterprogramme - Programmmoduln genannt - zu verfassen, die nun ihrerseits in weitere Moduln unterteilt werden können. Diese Verzweigung läßt sich durch ein Baumdiagramm darstellen.

Beispiel:

Ein Programm A zur Meßwertverarbeitung ruft eine Folge von drei Unterprogrammen auf:

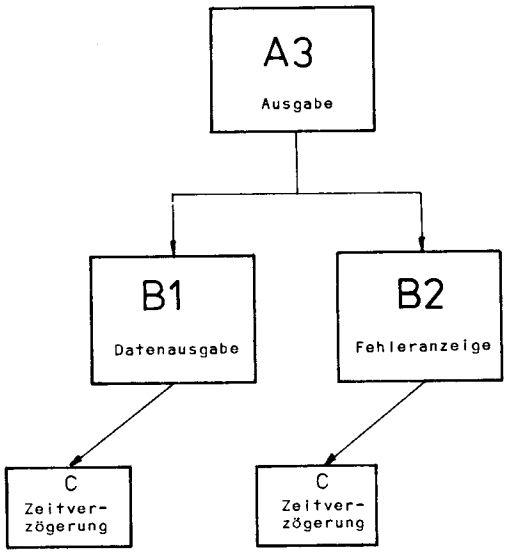
Unterprogramm A1 zur Meßwerteingabe
Unterprogramm A2 zur Meßwertberechnung und
Unterprogramm A3 zur Meßwertausgabe.

Folge von Unterprogrammen:



Das Unterprogramm A3 zur Meßwertausgabe ruft nun seinerseits entweder das Unterprogramm B1 zur Ausgabe gültiger Werte oder das Unterprogramm B2 zur Anzeige von Fehlern auf. Beide benötigen ein Unterprogramm C zur Zeitverzögerung.

Verzweigung bei Unterprogrammen:



Diese Baumdiagrammtechnik wurde von Jackson allgemein für die strukturierte Programmierung eingeführt, sie dient hier nur zur Darstellung von Unterprogrammen, da sie im Gegensatz zu den Struktogrammen (Programmblockplänen) nach Nassi-Shneiderman die Programmelemente als getrennte Blöcke darstellt. Dies unterstreicht die Tatsache, daß Unterprogramme als weitgehend selbständige Programmeinheiten angesehen werden können.

Damit ergibt sich für die grafischen Darstellungen folgende Einteilung:
Das Baumdiagramm beschreibt den Aufbau des Gesamtprogramms aus Haupt- und Unterprogrammen. Jedes Unterprogramm sollte eine Teilaufgabe erfüllen und nur so groß sein, daß es leicht überschaubar ist.
Der Programmblockplan (Struktogramm) beschreibt die Struktur des Programms ohne Rücksicht auf die Art der Ausführung. Man kann also für ein Programm zunächst einen groben Blockplan entwerfen und es dann mit einem Baumdiagramm in Unterprogramme unterteilen. Jedes Unterprogramm ist wieder durch einen Programmblockplan darstellbar.
Der Programmablaufplan zeigt die einzelnen Befehle. Er ist nicht dazu geeignet, größere Programme zu erläutern, kann aber sehr gut Einzelheiten beschreiben.

Bild 110 zeigt den Aufruf eines Unterprogramms. Gegenüber einem Sprungbefehl wird vorher der alte Stand des Befehlszählers in den Stapel gerettet. Der symbolische Unterprogrammname ist gleich der Adresse des ersten Befehls im Unterprogramm.

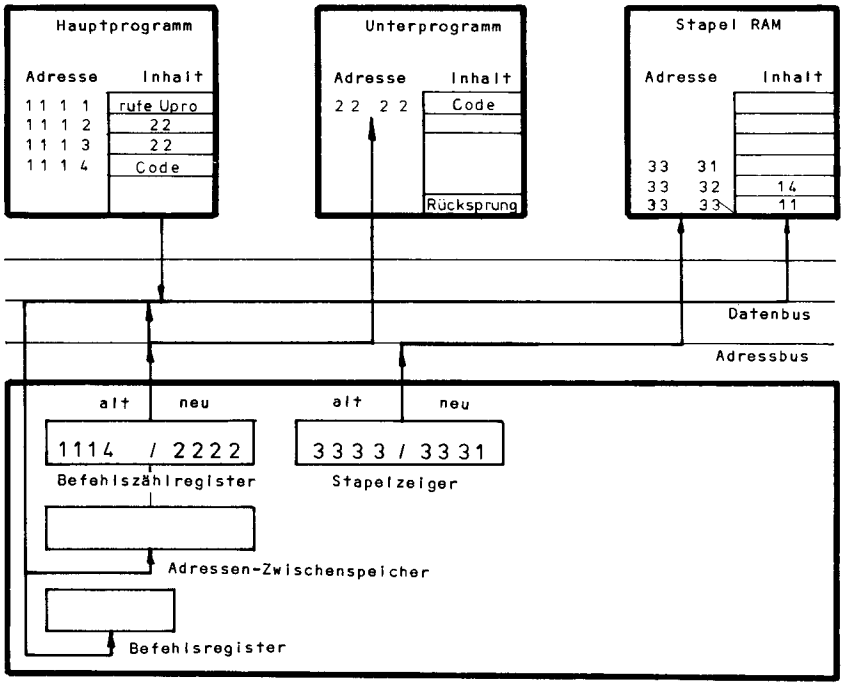


Bild 110: Aufruf eines Unterprogramms

Der Sprung in ein Unterprogramm erfolgt in mehreren Schritten:

a. Bei der Ausführung des Sprungbefehls gelangt die Anfangsadresse 2222 des Unterprogramms zunächst in den Adressen-Zwischenspeicher.

b. Der um 1 erhöhte Befehlszählerstand 1114 wird automatisch in den Stapel gerettet und der Stapelzeiger dabei von 3333 auf 3331 vermindert.
c. Die Anfangsadresse 2222 des Unterprogramms wird in den Befehlszähler gebracht und als neue Befehlsadresse ausgesendet.

Am Ende des Unterprogramms steht ein Befehl, der in das Hauptprogramm zurückspringt. Dies setzt seine Arbeit mit dem Befehl fort, der dem Unterprogrammaufruf folgt.

Bild 111 zeigt den Rücksprung vom Unterprogramm in das Hauptprogramm.

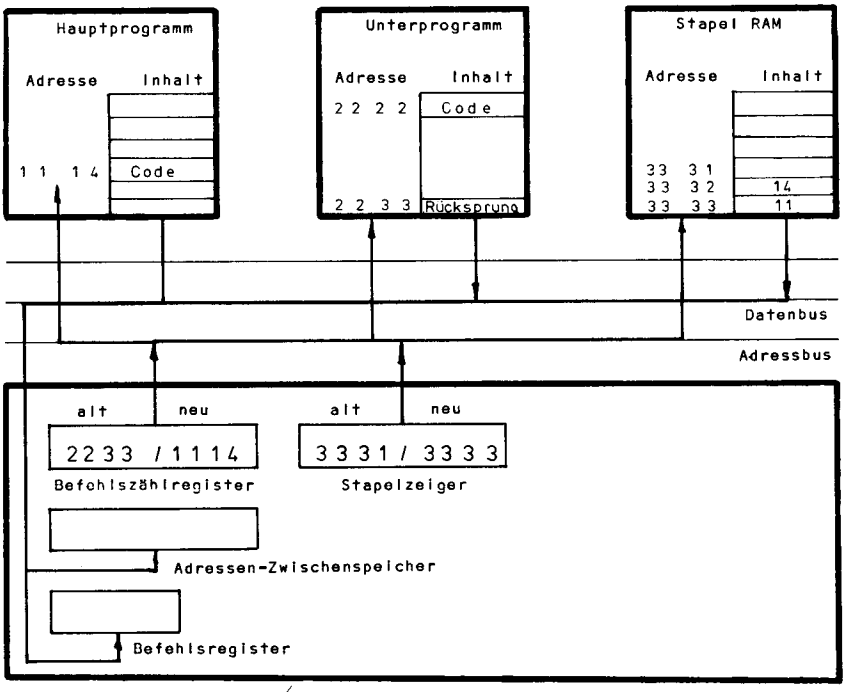


Bild 111: Rücksprung in das Hauptprogramm

Der Rücksprung in das Hauptprogramm erfolgt in mehreren Schritten:

a. Die Rücksprungadresse wird automatisch aus dem Stapel in den Adressen-Zwischenspeicher geholt. Dabei wird der Stapelzeiger um 2 erhöht.
b. Die Rücksprungadresse wird in den Befehlszähler gebracht und als neue Befehlsadresse ausgesendet.

Haupt- und Unterprogramm müssen sich gegenseitig Daten übergeben. Ein Verzögerungs-Unterprogramm benötigt z.B. vom Hauptprogramm die Verzögerungszeit und ein Rechenunterprogramm muß das Ergebnis an das Hauptprogramm zurückliefern. Dazu gibt es grundsätzlich folgende Möglichkeiten:

a. Übergabe von Daten, z.B. der Verzögerungszeit,
b. Übergabe von Datenadressen, z.B. der Adresse einer Speicherstelle, in der die Verzögerungszeit steht,
c. Übergabe der Adresse eines Speicherbereiches, in dem die Daten oder Adressen abgelegt sind,
d. Übergabe in einem Register des Mikroprozessors,
e. Übergabe im Stapel,
f. Übergabe in bestimmten Speicherstellen; vorzugsweise in den Bytes, die dem Unterprogramm-Sprungbefehl folgen,
g. Übergabe einer festen oder einer variablen Anzahl von Werten.

Man unterscheidet also

a. die Art (Inhalt, Adresse, Adresse eines Bereiches)
b. den Ort (Register, Stapel, Programmbytes) und
c. die Anzahl (fest oder variabel).

Die einfachste Art ist die Übergabe von Daten in den Registern des Mikroprozessors.
Da das Unterprogramm dieselben Register wie das Hauptprogramm benutzt, muß sichergestellt werden, daß keine Registerinhalte zerstört werden, die das Hauptprogramm nach dem Ablauf des Unterprogramms benötigt. Das Haupt- bzw. das Unterprogramm muß also eventuell Register retten und wieder zurückschreiben. Dies ist besonders wichtig bei Unterprogrammen, die zu einem anderen Zeitpunkt oder von einem anderen Programmierer erstellt werden.

Da die Rücksprungadresse im Stapel gerettet wird, können Unterprogramme beliebig geschachtelt werden. Ein Unterprogramm kann auch sich selbst aufrufen (Rekursion), muß aber dafür sorgen, daß die Kette der Aufrufe abgebrochen wird, wenn eine Abbruchbedingung erreicht ist.

7.2.1 Die Unterprogrammbefehle des KIM-1 KIM

Die folgenden Befehle bewirken einen Unterprogrammsprung bzw. einen Rücksprung in das Hauptprogramm.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV B DIZC	
JSR	Adresse	absolut	20	3	6		Unterprogrammsprung, Rücksprungadresse n. St.
RTS		code-eigen	60	1	6		Rücksprung, Adresse vom Stapel

JSR bedeutet Jump to SubRoutine = Springe in ein Unterprogramm, RTS bedeutet ReTurn from Subroutine = Rückkehr aus einem Unterprogramm.

Gegenüber den anderen Mikroprozessoren enthält der Stapel nicht die richtige Rücksprungadresse, sondern den um 1 verminderten Wert. Der Rücksprungbefehl muß deshalb diese Adresse um 1 erhöhen, bevor sie als neue Befehlsadresse ausgesendet wird. Für den Programmierer ist dies nur von Bedeutung, wenn er die Rücksprungadresse im Programm berechnen oder verändern will.

Beispiel:

Das folgende Testprogramm ruft ein Unterprogramm WARTE zur Zeitverzögerung auf. Das Hauptprogramm übergibt die Zeit im X-Register.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0310	
0310	A2	MAX	LDX	#$FF
	FF			
0312	20		JSR	WARTE
	30			
0314	03			
0315	4C		JMP	MAX
	10			
	03			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0330	
0330	F0	WARTE	BEQ	RUECK
	04			
0332	CA		DEX	
0333	4C		JMP	WARTE
	30			
	03			
0336	60	RUECK	RTS	

Vor dem Unterprogrammaufruf:

Stapelzeiger	Stapel	Adresse	Inhalt
FF →		01FF	beliebig

Nach dem Unterprogrammaufruf

Stapelzeiger	Stapel	Adresse	Inhalt
FD →		01FD	beliebig
		01FE	14
		01FF	03

Nach dem Rücksprung

Stapelzeiger	Stapel	Adresse	Inhalt
FF →		01FF	03

Bei jedem Unterprogrammaufruf wird zuerst das höherwertige und dann das niederwertige Byte der Rücksprungadresse in den Stapel gerettet. Beim Aufruf eines Unterprogramms kann man Adressen bzw. Werte in den auf den Aufrufbefehl folgenden Bytes im Hauptprogramm ablegen. Das Unterprogramm muß in diesen Fällen die im Stapel liegende Rücksprungadresse um die Anzahl der hinter dem Unterprogrammaufruf abgelegten Bytes erhöhen, um an die richtige Stelle in das Hauptprogramm zurückkehren zu können.

Das folgende Testprogramm ruft ein Unterprogramm MAXI auf, das in eine Speicherstelle einen konstanten Wert bringt. Die Adresse der Speicherstelle und die Konstante werden hinter dem Befehl JSR MAXI im Hauptprogramm abgelegt. Das Unterprogramm übernimmt die Konstante und speichert sie mit indirekter Adressierung in die Speicherstelle, deren Adresse im Hauptprogramm abgelegt wurde. Da alle indirekten Adressen in Seite Null liegen müssen, gilt das Unterprogramm nur für den Fall, daß das Hauptprogramm in Seite Null liegt. Ist dies nicht möglich, so müssen für die indirekte Adresse zwei Hilfs-Speicherstellen in Seite Null reserviert werden. Damit ist das Unterprogramm nicht mehr unterbrechungsfest (re-entrant).

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0030	
0030	A2	START	LDX	#$7F
	7F			
0032	9A		TXS	
0033	20		JSR	MAXI
	00			
	03			
0036	01		.WORD	$0001
	00			
0038	FF		.BYTE	$FF
0039	4C	OTTO	JMP	OTTO
	39			
	00			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			*=$0300	
0300	BA	MAXI	TSX	
0301	8A		TXA	
0302	A8		TAY	
0303	BE		LDX	$0101,Y
	01			
	01			
0306	BD		LDA	$0003,X
	03			
	00			
0309	81		STA	($01,X)
	01			
030B	68		PLA	
030C	18		CLC	
030D	69		ADC	#$03
	03			
030F	AA		TAX	
0310	68		PLA	
0311	69		ADC	#$00
	00			
0313	48		PHA	
0314	8A		TXA	
0315	48		PHA	
0316	60		RTS	
			.END	

7.2.2 Die Unterprogrammbefehle des 6800

6800

Die folgenden Befehle bewirken einen Unterprogrammsprung bzw. einen Rücksprung in das Hauptprogramm.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HI	NZVC	
JSR	Adresse	absolut	BD	3	9			Unterprogrammsprung, Adresse extended
JSR	konst,X	indiziert	AD	2	8			Unterprogrammsprung, Adresse indexed
BSR	Adresse	relativ	8D	2	8			Unterprogrammsprung, Adresse relativ
RTS		code-eigen	39	1	5			Rücksprung in das Hauptprogramm

JSR bedeutet Jump to SubRoutine = springe in ein Unterprogramm.
BSR bedeutet Branch to SubRoutine = verzweige in ein Unterprogramm.
RTS bedeutet ReTurn from Subroutine = Rücksprung aus einem Unterprogramm.

Beispiel:

Das folgende Testprogramm ruft ein Unterprogramm WARTE zur Zeitverzögerung auf. Das Hauptprogramm übergibt die Verzögerungszeit im X-Register.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0180
0180	CE	MAX	LDX	#$1000
	10			
	00			
0183	AD		JSR	WARTE
	01			
	90			
0186	7E		JMP	MAX
	01			
	80			

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	$0190
0190	27	WARTE	BEQ	RUECK
	04			
0192	09		DEX	
0193	7E		JMP	WARTE
	01			
	90			
0196	39	RUECK	RTS	
			END	

Vor dem Unterprogrammaufruf

Stapelzeiger	Stapel	Adresse	Inhalt
00FF ⟶		00FF	beliebig

Nach dem Unterprogrammaufruf

Stapelzeiger	Stapel	Adresse	Inhalt
00FD ⟶		00FD	beliebig
		00FE	01
		00FF	86

Nach dem Rücksprung

Stapelzeiger	Stapel	Adresse	Inhalt
00FF ⟶		00FF	86

Bei jedem Unterprogrammaufruf wird zuerst das niederwertige und dann das höherwertige Byte der Rücksprungadresse in den Stapel gerettet. Legt man bei einem Unterprogrammaufruf hinter dem Befehl Adressen und Werte ab, so muß das Unterprogramm

a. mit Hilfe der indirekten Adressierung die Adressen bzw. Werte verarbeiten und
b. die Rücksprungadresse um die Zahl der hinter dem Sprungbefehl benutzten Bytes erhöhen, um zum nächsten Befehl des Hauptprogramms zurückkehren zu können.

Das folgende Testprogramm übergibt einem Unterprogramm die Adresse einer Speicherstelle und eine Konstante. Beide liegen hinter dem Sprungbefehl. Das Unterprogramm bringt die Konstante in die angegebene Speicherstelle und springt zum nächsten Befehl des Hauptprogramms zurück.

Hauptprogramm

Hexadezimal Adresse	Hexadezimal Inhalt	Symbolisch Name	Symbolisch Befehl	Symbolisch Operand
			ORG	$0180
0180	8E	START	LDS	#$01FF
	01			
	FF			
0183	BD		JSR	MAXI
	01			
	90			
0186	00		FDB	$0001
	01			
0188	FF		FCB	$FF
0189	20	OTTO	BRA	OTTO
	FE			

Unterprogramm

Hexadezimal Adresse	Hexadezimal Inhalt	Symbolisch Name	Symbolisch Befehl	Symbolisch Operand
			ORG	$0190
0190	30	MAXI	TSX	
0191	EE		LDX	00,X
	00			
0193	A6		LDA A	02,X
	02			
0195	EE		LDX	00,X
	00			
0197	A7		STA A	00,X
	00			
0199	32		PUL A	
019A	33		PUL B	
019B	CB		ADD B	#$03
	03			
019D	89		ADC A	#$00
	00			
019F	37		PSH B	
01A0	36		PSH A	
01A1	39		RTS	
			END	

7.2.3 Die Unterprogrammbefehle des 8080/8085

Die folgenden Befehle bewirken einen Unterprogrammsprung bzw. einen Rücksprung in das Hauptprogramm.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
CALL	Adresse	absolut	CD	3	17			Unbedingter Unterprogrammsprung
CC	Adresse	absolut	DC	3	11/17			Unterprogrammsprung bei C = 1
CNC	Adresse	absolut	D4	3	11/17			Unterprogrammsprung bei C = 0
CZ	Adresse	absolut	CC	3	11/17			Unterprogrammsprung bei Z = 1
CNZ	Adresse	absolut	C4	3	11/17			Unterprogrammsprung bei Z = 0
CM	Adresse	absolut	FC	3	11/17			Unterprogrammsprung bei N = 1
CP	Adresse	absolut	F4	3	11/17			Unterprogrammsprung bei N = 0
CPE	Adresse	absolut	EC	3	11/17			Unterprogrammsprung bei P = 1
CPO	Adresse	absolut	E4	3	11/17			Unterprogrammsprung bei P = 0

CALL bedeutet rufen. Die Kennbuchstaben der übrigen Befehle enthalten Aufrufbedingung.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
RET		code-eigen	C9	1	10			Unbedingter Rücksprung ins Hauptprogramm
RC		code-eigen	D8	1	5/11			Rücksprung bei C = 1
RNC		code-eigen	D0	1	5/11			Rücksprung bei C = 0
RZ		code-eigen	C8	1	5/11			Rücksprung bei Z = 1
RNZ		code-eigen	C0	1	5/11			Rücksprung bei Z = 0
RM		code-eigen	F8	1	5/11			Rücksprung bei N = 1
RP		code-eigen	F0	1	5/11			Rücksprung bei N = 0
RPE		code-eigen	E8	1	5/11			Rücksprung bei P = 1
RPO		code-eigen	E0	1	5/11			Rücksprung bei P = 0

RET bedeutet RETurn = zurückspringen. Die Kennbuchstaben der übrigen Befehle enthalten die Rücksprungbedingung.

Die erste Zykluszeit gilt, wenn die Sprungbedingung nicht erfüllt ist und der Sprung nicht ausgeführt wird, die zweite für den Fall, daß gesprungen wird.

Beispiel:

Das folgende Testprogramm ruft ein Unterprogramm WARTE zur Zeitverzögerung auf. Das Hauptprogramm übergibt die Verzögerungszeit im D-Register.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2880H
2880	16	MAX	MVI	D,0FFH
	FF			
2882	CD		CALL	WARTE
	90			
	28			
2885	C3		JMP	MAX
	80			
	28			
			END	

Unterprogramm

Hexadezimal Adresse	Inhalt	Name	Symbolisch Befehl	Operand
			ORG	2890H
2890	C8	WARTE	RZ	
2891	15		DCR	D
2892	C3		JMP	WARTE
	90			
	28			
			END	

Vor dem Unterprogrammaufruf

Stapelzeiger	Stapel	Adresse	Inhalt
20E3 ⟶		20E3	beliebig

Nach dem Unterprogrammaufruf

Stapelzeiger	Stapel	Adresse	Inhalt
20E1 ⟶		20E1	85
		20E2	28
		20E3	beliebig

Nach dem Rücksprung

Stapelzeiger	Stapel	Adresse	Inhalt
20E3 ⟶		20E3	beliebig

Bei jedem Unterprogrammaufruf wird zuerst das höherwertige und dann das niederwertige Byte der Rücksprungadresse in den Stapel gebracht. Legt man im Hauptprogramm hinter dem Sprungbefehl Adressen und Werte ab, so muß das Unterprogramm

a. mit Hilfe der indirekten Adressierung die Adressen bzw. Werte verarbeiten und
b. die Rücksprungadresse um die Zahl der hinter dem Sprungbefehl benutzten Bytes erhöhen, um zum nächsten Befehl des Hauptprogramms zurückkehren zu können.

Das folgende Testprogramm übergibt einem Unterprogramm die Adresse einer Speicherstelle und eine Konstante. Das Unterprogramm bringt die Konstante in die adressierte Speicherstelle und springt zum nächsten Befehl des Hauptprogramms zurück.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	31	START	LXI	SP,2900H
	00			
	29			
2883	CD		CALL	BUBI
	90			
	28			
2886	00		DW	2000H
	20			
2888	FF		DB	0FFH
2889	C3	OTTO	JMP	OTTO
	89			
	28			

Unterprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2890H
2890	E3	BUBI	XTHL	
2891	4E		MOV	C,M
2892	23		INX	H
2893	46		MOV	B,M
2894	23		INX	H
2895	7E		MOV	A,M
2896	23		INX	H
2897	02		STAX	B
2898	E3		XTHL	
2899	C9		RET	
			END	

7.3 Programmunterbrechungen durch äußere Signale (Interrupt)

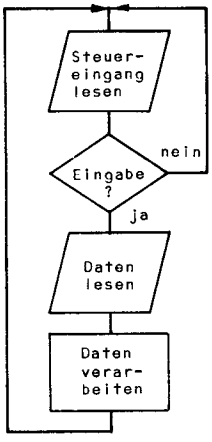
Bei der Arbeit mit äußeren Geräten und Schaltungen unterscheidet man Steuersignale und Daten.
Daten sind Meßwerte, Zahlen oder Zeichen.
Steuersignale melden, daß Daten am Eingang bzw. Ausgang der Ein/Ausgaberegister anliegen.

In den bisher gezeigten Beispielen wurde die Eingabe dauernd nach Steuersignalen oder Daten abgefragt (Polling).

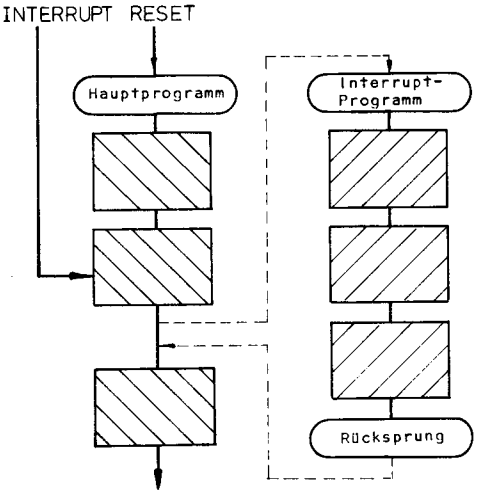
Läßt man dagegen die Steuersignale direkt auf die Interrupt-Eingänge des Mikroprozessors und damit auf die Ablaufsteuerung wirken, so kann man damit das laufende Programm unterbrechen und ein Programm zum Lesen der Daten starten. Das Programm zum Abfragen der Steuereingänge entfällt; das Leseprogramm arbeitet nur, wenn tatsächlich Daten anliegen. In der Zwischenzeit kann ein weniger wichtiges Programm (Hintergrundprogramm) arbeiten. Diese Programmunterbrechung kann zu jedem Zeitpunkt erfolgen. Der laufende Befehl wird noch zuendegeführt, der Befehlszähler in den Stapel gerettet und ein Unterbrechungsprogramm gestartet. Anschließend kann man durch einen Rücksprungbefehl wieder in das unterbrochene Programm zurückkehren. Diese Programmunterbrechung (Interrupt) wirkt wie ein Unterprogramm, das nicht vom Hauptprogramm, sondern über eine äußere Signalleitung aufgerufen wird. Über besondere Befehle läßt sich die Programmunterbrechung auch sperren.

Nach dem Einschalten der Versorgungsspannung sind alle Register des Mikroprozessors, alle RAM-Speicher und alle Ein/Ausgabe-Register in einem nicht definierten - zufälligen - Zustand. Das Programm liegt unzerstörbar in einem Nur-Lese-Speicher (ROM). Es wird durch das RESET-Signal gestartet.

Abfrage



Interrupt



7.3.1 Die Interrupt-Möglichkeiten des KIM-1

Der Mikroprozessor 6502 hat drei Interrupt-Leitungen:

a. Wird die RESET-Leitung von 0 auf 1 gebracht, so sendet der Mikroprozessor nacheinander die Adressen FFFC und FFFD aus und lädt den Inhalt dieser beiden Bytes in das Befehlszählregister.
Die Speicherstelle FFFC enthält das niederwertige Byte der Startadresse.
Die Speicherstelle FFFD enthält das höherwertige Byte der Startadresse.
Während der Anlaufphase wird das I-Bit des Bedingungsregisters auf 1 gesetzt, d.h. der Interrupt IRQ ist gesperrt und muß erst durch das Programm freigegeben werden.

b. Wird die NMI-Leitung von 1 auf 0 gebracht, so werden
- der laufende Befehl zuendegeführt,
- das höherwertige und das niederwertige Byte des Befehlszählregisters sowie das Bedingungsregister in den Stapel gerettet und der Stapelzeiger um 3 vermindert,
- nacheinander die Adressen FFFA und FFFB ausgesendet und der Inhalt der beiden Bytes in das Befehlszählregister gebracht und
- das I-Bit des Bedingungsregisters 1 gesetzt (IRQ-Interrupt gesperrt).

Die Speicherstelle FFFA enthält das niederwertige Byte der Startadresse des Unterbrechungsprogramms,
die Speicherstelle FFFB enthält das höherwertige Byte der Startadresse des Unterbrechungsprogramms.
Der NMI-Interrupt kann nicht gesperrt werden (Nicht Maskierbarer Interrupt). Er wirkt bei einem Wechsel der Leitung von 1 auf 0.

c. Wird die IRQ-Leitung auf 0 gebracht, so werden
- der laufende Befehl zuendegeführt,
- das höherwertige und das niederwertige Byte des Befehlszählregisters sowie das Bedingungsregister in den Stapel gerettet und der Stapelzeiger um 3 vermindert,
- nacheinander die Adressen FFFE und FFFF ausgesendet und der Inhalt der beiden Bytes in das Befehlszählregister gebracht und
- das I-Bit des Bedingungsregisters 1 gesetzt (IRQ-Interrupt gesperrt).

Die Speicherstelle FFFE enthält das niederwertige Byte der Startadresse des Unterbrechungsprogramms,
die Speicherstelle FFFF enthält das höherwertige Byte der Startadresse des Unterbrechungsprogramms.
Der IRQ-Interrupt ist bei I=1 des Bedingungsregisters unwirksam. Er wirkt solange, wie die Leitung auf 0 liegt.

Anordnung der Interrupt-Adressen:

FFFA	niederes Byte der NMI-Startadresse
FFFB	höheres Byte der NMI-Startadresse
FFFC	niederes Byte der RESET-Startadresse
FFFD	höheres Byte der RESET-Startadresse
FFFE	niederes Byte der IRQ- bzw. BRK-Startadresse
FFFF	höheres Byte der IRQ- bzw. BRK-Startadresse

Anordnung der geretteten Register im Stapel:

Stapelzeiger neu:⟶
Bedingungsregister
niederes Byte der Rücksprungadresse
Stapelzeiger alt:⟶höheres Byte der Rücksprungadresse

Mit Hilfe des BRK-Befehls ist es möglich, vom Programm aus den IRQ-Interrupt auszulösen.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
BRK		code-eigen	00	1	7		1			IRQ-Interrupt des Programms

BRK bedeutet BReaK = veranlasse Programmunterbrechung.

Um diesen Sofware-Interrupt vom Hardware-Interrupt unterscheiden zu können, setzt der BRK-Befehl das B-Bit des Bedingungsregisters auf 1; beim Hardware-Interrupt ist es 0.

Der BRK-Befehl legt nicht wie der Hardware-Interrupt die Rücksprungadresse, sondern die um 1 erhöhte Rücksprungadresse im Stapel ab. Beim Rücksprung mit dem Befehl RTI würde also das auf den Befehl BRK folgende Byte übersprungen werden. Am Besten läßt man auf den BRK-Befehl immer einen NOP-Befehl folgen.

Am Ende der Interrupt-Programme steht der Befehl RTI , der ähnlich wie der Rücksprungbefehl eines Unterprogramms den alten Befehlszählerstand und das Bedingungsregister aus dem Stapel holt und damit das unterbrochene Programm fortsetzt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
RTI		code-eigen	40	1	6	vom Stapel				Rücksprung aus Unterbrechungsprogramm

RTI bedeutet ReTurn from Interrupt = Rückkehr aus einer Programmunterbrechung.

Die beiden folgenden Befehle schalten die Sperre für den IRQ-Interrupt ein bzw. aus.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NV	B	DI	ZC	
CLI		code-eigen	58	1	2			0		IRQ-Interrupt erlaubt
SEI		code-eigen	78	1	2			1		IRQ-Interrupt gesperrt

CLI bedeutet CLear Interrupt disable bit = lösche Interruptsperre.
SEI bedeutet SEt Interrupt disable bit = setze Interruptsperre.

Während der Bearbeitung des IRQ-Interrupts ist das I-Bit des Bedingungsregisters 1, d.h. der Prozessor ist für weitere IRQ-Interrupts gesperrt. Der Befehl CLI hebt diese Sperre auf. Damit sind verschachtelte Interrupts möglich.

Treten ein NMI- und ein IRQ-Interrupt gleichzeitig auf, so wird der NMI-Interrupt bevorzugt behandelt.

Bild 112 zeigt ein Beispiel für die Interrupt-Adressen des KIM-1.

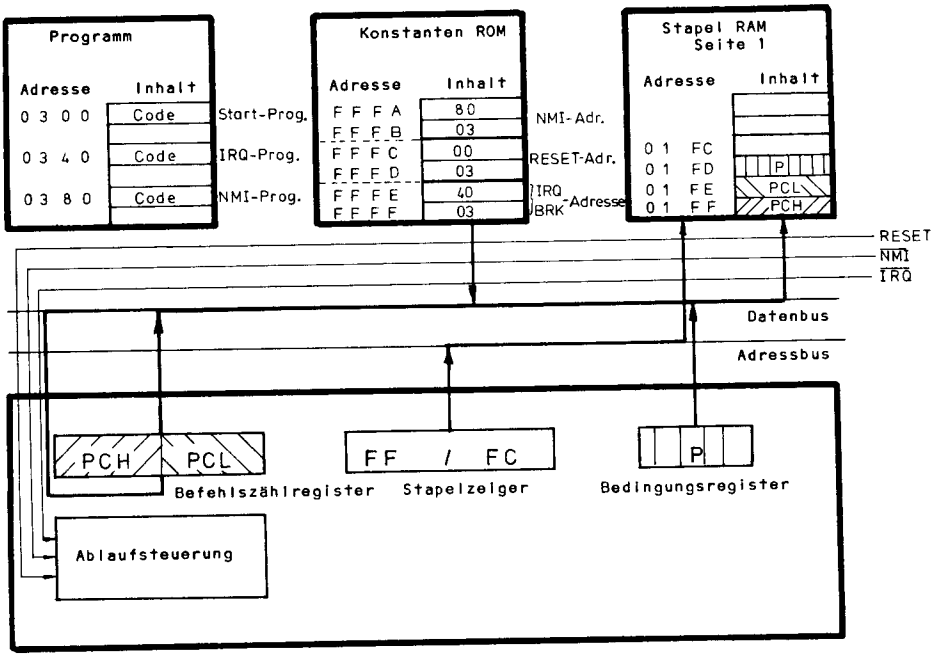


Bild 112: Interrupt des KIM-1

Die Startadresse des Hauptprogramms ist 0300.
Die Startadresse des IRQ-Interruptprogramms ist 0340.
Die Startadresse des NMI-Interruptprogramms ist 0380.

Beispiele:

Die folgenden Testprogramme untersuchen die Interrupt-Möglichkeiten mit Hilfe einer Schaltung nach Bild 48. Durch den Schaltungsaufbau des KIM-1 und wegen der Kontrolle des Benutzerprogramms durch ein Betriebsprogramm liegen die Startadressen der Unterbrechungsprogramme im Bereich von 17FA bis 17FF statt zwischen FFFA und FFFF.

a. Der BRK-Befehl

Die folgenden Programme testen den BRK-Befehl.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		RAU	=$1701	
			*=$0300	
0300	A9	START	LDA	#$FF
	FF			
0302	8D		STA	RAU
	01			
	17			
0305	00		BRK	
0306	EA		NOP	
0307	4C	OTTO	JMP	OTTO
	07			
	03			

Unterbrechungsprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	=$1700	
			*=$0380	
0380	A9		LDA	#$AA
	AA			
0382	8D		STA	AUS
	00			
	17			
0385	40		RTI	
			.END	

Vor dem Start des Hauptprogramms muß die Speicherstelle 17FE mit 80 und die Speicherstelle 17FF mit 03 geladen werden. Dies ist die Startadresse des Unterbrechungsprogramms. Beim Aufruf des Unterbrechungsprogramms erscheint die Bitkombination 101010 10 an der Ausgabe. Das Hauptprogramm muß durch die STOP-Taste angehalten werden. Der Befehlszählerstand ist 0307.

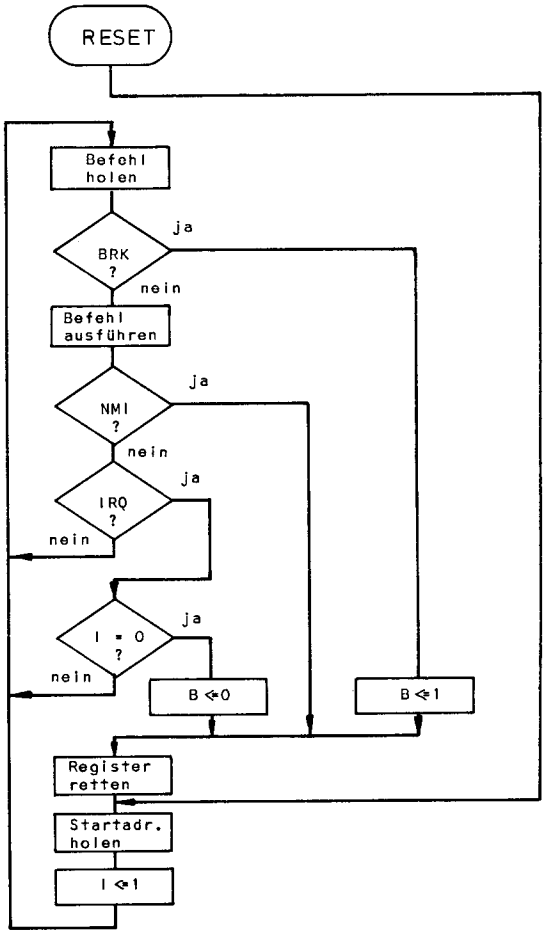
b. Der IRQ-Interrupt

Im Hauptprogramm wird der Befehl BRK durch einen CLI-Befehl mit dem Funktionscode 58 ersetzt, der eine eventuell vorhandene Interrupt-Sperre aufhebt. Die Startadresse des Unterbrechungsprogramms bleibt wie beim BRK-Befehl in den Speicherstellen 17FE und 17FF. Bringt man die Leitung IRQ (Stift E4 des Expansionssteckers) auf das Potential 0 Volt, so wird die Bitkombination 10101010 angezeigt.Durch die STOP-Taste wird das Programm beim Befehlszählerstand 0307 abgebrochen.

c. Der NMI-Interrupt

Im Hauptprogramm wird der Befehl BRK durch einen NOP-Befehl mit dem Funktionscode EA ersetzt. Die Speicherstelle 17FA wird mit 80 und die Speicherstelle 17FB wird mit 03 geladen. Dies ist die Startadresse des Unterbrechungsprogramms. Bringt man nun die Leitung NMI (Stift E6 des Expansionssteckers) auf das Potential 0 Volt, so wird die Bitkombination 10101010 angezeigt. Da die STOP-Taste auch mit der NMI-Leitung arbeitet und daher dasselbe Unterbrechungsprogramm startet, kann das Programm nur mit RESET abgebrochen werden.

Der folgende Programmablaufplan zeigt schematisch die Wirkung der verschiedenen Unterbrechungsmöglichkeiten.



7.3.2 Die Interrupt-Möglichkeiten des 6800 **6800**

Der Mikroprozessor 6800 hat folgende Interrupt-Leitungen:

a. Wird die RESET-Leitung von 1 auf 0 gebracht, so sendet der Mikroprozessor nacheinander die Adressen FFFE und FFFF aus und lädt den Inhalt dieser beiden Bytes in das Befehlszählregister. Die Speicherstelle FFFE enthält das höherwertige Byte der Startadresse,
die Speicherstelle FFFF enthält das niederwertige Byte der Startadresse.

b. Wird die NMI-Leitung von 1 auf 0 gebracht, so werden
- der laufende Befehl zuendegeführt,
- alle Register des Mikroprozessors in den Stapel gerettet,
- nacheinander die Adressen FFFC und FFFD ausgesendet und der Inhalt dieser beiden Bytes in das Befehlszählregister gebracht.

Die Speicherstelle FFFC enthält das höherwertige Byte der Startadresse des Unterbrechungsprogramms,
die Speicherstelle FFFD enthält das niederwertige Byte der Startadresse des Unterbrechungsprogramms.

c. Liegt die IRQ-Leitung auf dem Potential 0 Volt, so werden
   - der laufende Befehl zuendegeführt,
   - alle Register des Mikroprozessors in den Stapel gerettet,
   - das I-Bit des Bedingungsregisters auf 1 gesetzt, um den Eingang für weitere Interrupts zu sperren,
   - nacheinander die Adressen FFF8 und FFF9 ausgesendet und der Inhalt dieser beiden Bytes in das Befehlszählregister gebracht.

   Die Speicherstelle FFF8 enthält das höherwertige Byte der Startadresse des Unterbrechungsprogramms,
   die Speicherstelle FFF9 enthält das niederwertige Byte der Startadresse des Unterbrechungsprogramms.
   Der IRQ-Interrupt ist bei I=1 des Bedingungsregisters gesperrt.

d. Der Befehl SWI (SoftWare-Interrupt) löst einen programmgesteuerten Interrupt aus.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
SWI		code-eigen	3F	1	12	1	vom Programm ausgelöster Interrupt

SWI bedeutet SoftWare - Interrupt = programmgesteuerte Unterbrechung.

Wie bei einem äußeren Interrupt erhöht der Prozessor den Befehlszähler um 1, rettet alle Register in den Stapel, setzt das I-Bit auf 1, sendet nacheinander die Adressen FFFA und FFFB aus und bringt den Inhalt dieser beiden Bytes in das Befehlszählregister.
Die Speicherstelle FFFA enthält das höherwertige Byte der Startadresse des Unterbrechungsprogramms,
die Speicherstelle FFFB enthält das niederwertige Byte der Startadresse des Unterbrechungsprogramms.

e. Der Befehl WAI (WArte auf Interrupt) bringt den Mikroprozessor in einen Wartezustand, in dem er keine Befehle mehr ausführt, sondern nur noch auf einen Interrupt wartet. Der Befehlszähler wird um 1 weitergezählt und zusammen mit den anderen Registern in den Stapel gerettet.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
WAI		code-eigen	3E	1	9		Warte auf Interrupt

Ist das I-Bit des Bedingungsregisters 1, so kann die Arbeit des Mikroprozessors nur mit einem RESET oder NMI-Interrupt fortgesetzt werden.

Die beiden folgenden Befehle schalten die Sperre für den IRQ-Interrupt ein bzw. aus.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
CLI		code-eigen	0E	1	2	0	IRQ-Interrupt erlaubt
SEI		code-eigen	0F	1	2	1	IRQ-Interrupt gesperrt

CLI bedeutet CLear Interrupt mask = lösche Interrupt-Sperre.
SEI bedeutet SEt Interrupt mask = setze Interrupt-Sperre.

Der NMI-Interrupt wird bevorzugt behandelt und kann nicht gesperrt werden.

Am Ende der Interrupt-Programme steht der Befehl RTI, der wie ein Rücksprungbefehl eines Unterprogramms den alten Registerstand vor der Unterbrechung wiederherstellt und damit das unterbrochene Programm fortsetzt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen --HINZVC	
RTI		code-eigen	3B	1	10	vom Stapel	Rücksprung aus Unterbrechungsprogramm

RTI bedeutet ReTurn from Interrupt = Rückkehr aus einer Programmunterbrechung.

Anordnung der Interrupt-Adressen:

FFF8 höherwertiges Byte der IRQ-Startadresse
FFF9 niederwertiges Byte der IRQ-Startadresse

FFFA höherwertiges Byte der SWI-Startadresse
FFFB niederwertiges Byte der SWI-Startadresse

FFFC höherwertiges Byte der NMI-Startadresse
FFFD niederwertiges Byte der NMI-Startadresse

FFFE höherwertiges Byte der RESET-Startadresse
FFFF niederwertiges Byte der RESET-Startadresse

Anordnung der geretteten Register im Stapel:

Stapelzeiger neu:→ Bedingungsregister
Akkumulator B
Akkumulator A
höherwertiges Byte des X-Registers
niederwertiges Byte des X-Registers
höherwertiges Byte der Rücksprungadresse
Stapelzeiger alt:→ niederwertiges Byte der Rücksprungadresse

Bild 113 zeigt ein Beispiel für die Anordnung der Interrupt-Adressen.

Die Startadresse des Hauptprogramms ist 0100.
Die Startadresse des NMI-Interruptprogramms ist 0140.
Die Startadresse des SWI-Interruptprogramms ist 0180.
Die Startadresse des IRQ-Interruptprogramms ist 01A0.

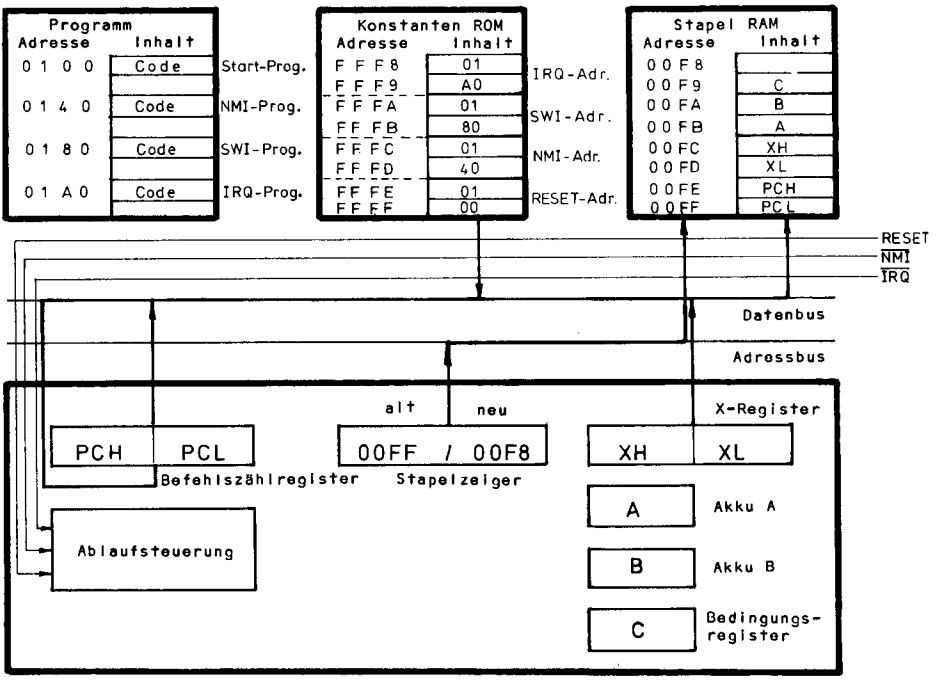


Bild 113: Interrupt des 6800

Beispiele:

Die folgenden Testprogramme untersuchen die Interrupt-Möglichkeiten mit Hilfe einer Schaltung nach Bild 48. Die Ausführung der Programme hängt vom Aufbau des verwendeten Entwicklungssystems ab, die Beispiele gelten für den Bausatz MEK 6800 D2.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	EQU	$8004
		KONA	EQU	$8005
			ORG	$0180
0180	86	START	LDA A	#$FF
	FF			
0182	B7		STA A	AUS
	80			
	04			
0185	86		LDA A	#$04
	04			
0187	B7		STA A	KONA
	80			
	05			
018A	0E		CLI	
018B	01		NOP	
018C	20	OTTO	BRA	OTTO
	FE			
			END	

Unterbrechungsprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
		AUS	EQU	$8004
			ORG	$0190
0190	C6		LDA B	#$AA
	AA			
0192	F7		STA B	AUS
	80			
	04			
0195	3B		RTI	
			END	

a. Der IRQ-Interrupt

Die Startadresse des Unterbrechungsprogramms ist in den Speicherstellen A000 und A001 abzulegen, da das Benutzerprogramm von einem Betriebsprogramm kontrolliert wird.

A000 erhält den Wert 01
A001 erhält den Wert 90

Legt man die IRQ-Leitung auf das Potential 0 Volt, so erscheint an der Ausgabe die Bitkombination 10101010.

b. Der NMI-Interrupt

Die Startadresse des Unterbrechungsprogramms ist in den Speicherstellen A006 und A007 abzulegen.

A006 erhält den Wert 01
A007 erhält den Wert 90

Legt man die NMI-Leitung auf das Potential 0 Volt, so erscheint an der Ausgabe die Bitkombination 10101010.

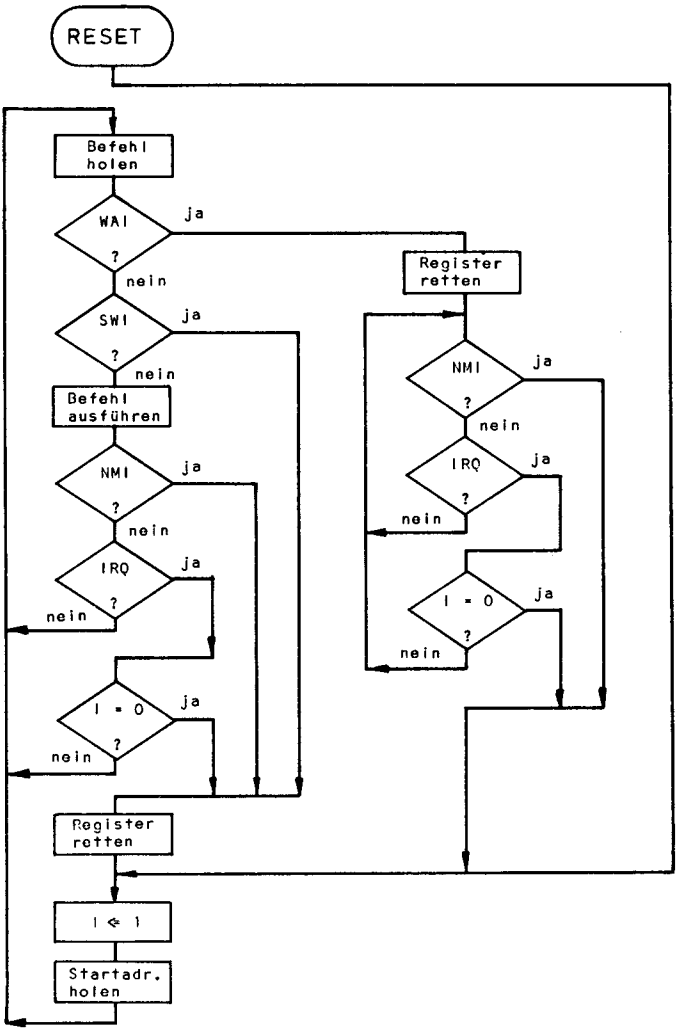
c. Der SWI-Interrupt

läßt sich nicht testen, da er für die Programmverfolgung durch das Betriebssystem verwendet wird. Baut man ihn z.B. anstelle des NOP-Befehls bei der Adresse 018B ein, so hält das Programm an dieser Stelle an.

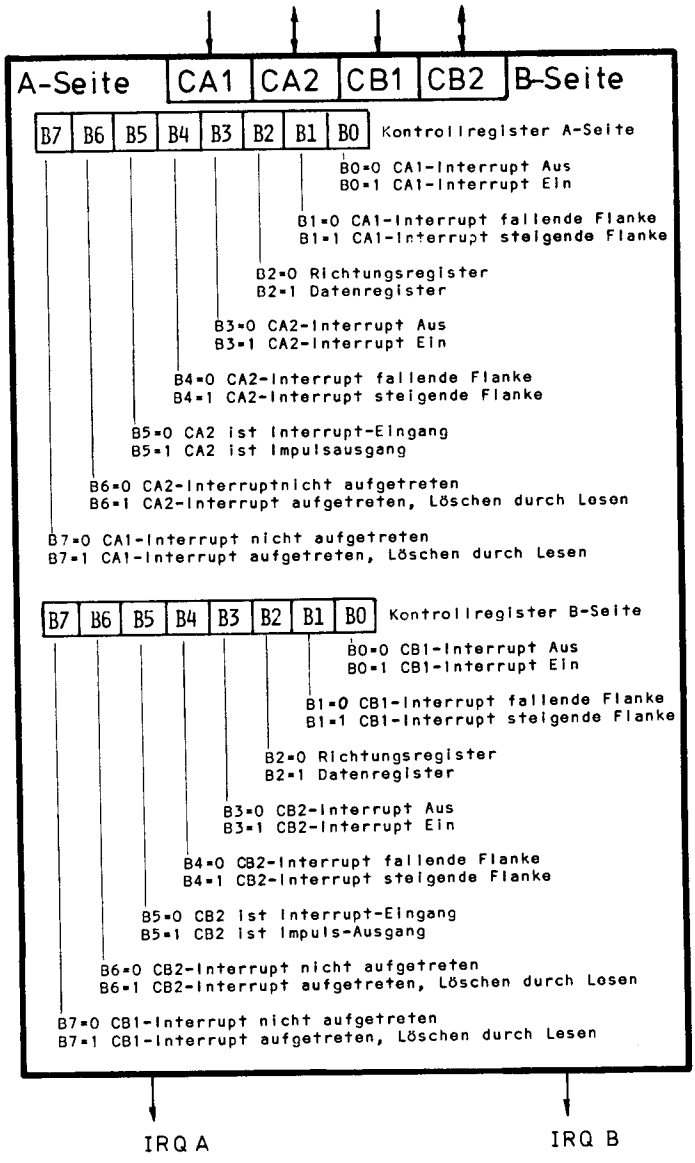
d. Der WAI-Befehl

kann anstelle des NOP-Befehls bei der Adresse 018B eingebaut und mit NMI- bzw. IRQ-Signalen getestet werden.

Der folgende Programmablaufplan zeigt schematisch, wie die verschiedenen Interruptmöglichkeiten verarbeitet werden.



Arbeiten mehrere Geräte auf eine Interrupt-Leitung, so können sie über eine Parallelschnittstelle 6820 leicht vom Programm kontrolliert werden. Jede der beiden Seiten schaltet zwei Eingangsleitungen CA1, CA2, CB1 und CB2 auf die Ausgänge IRQA bzw. IRQB, die auf die Interrupt-Leitungen führen. Über bestimmte Bits der beiden Kontrollregister kann man
den Interrupt sperren oder freigeben,
die auslösende Flanke des Signals bestimmen,
den Baustein auf einen Interrupt abfragen und
den Anschluß CA2 bzw. CB2 zum Eingang oder Ausgang erklären.



Interrupt-Schaltung der Parallelschnittstelle 6820

7.3.3 Die Interrupt-Möglichkeiten des 8080

8080

Der Mikroprozessor Intel 8080 hat einen RESET- und einen INT-Eingang.

a. Ein 1-Signal auf der RESET-Leitung setzt den Befehlszähler auf 0000 und startet damit das Hauptprogramm, das mit der Anfangsadresse 0000 beginnen muß.

b. Ein 1-Signal auf dem Interrupt-Eingang INT bewirkt, daß zunächst der laufende Befehl zuendegeführt wird. Das INTE-Flip-Flop wird auf 0 gesetzt, sodaß der Prozessor für weitere Interrupts gesperrt ist. Durch ein Signal auf der Ausgangsleitung INTE werden die Systemsteuerung bzw. die Eingabebausteine aufgefordert, über den Datenbus ein Byte zu senden, das vom Prozessor als Funktionscode eines neuen Befehls verarbeitet wird.
Besonders dafür geeignet sind die Funktionscode der folgenden 1-Byte-Befehle, die wie beim Unterprogrammaufruf den Stand des Befehlszählers in den Stapel retten.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C		
RST	0	code-eigen	C7	1	11			Befehlszähler nach Stapel, Startadresse 0000
RST	1	code-eigen	CF	1	11			Befehlszähler nach Stapel, Startadresse 0008
RST	2	code-eigen	D7	1	11			Befehlszähler nach Stapel, Startadresse 0010
RST	3	code-eigen	DF	1	11			Befehlszähler nach Stapel, Startadresse 0018
RST	4	code-eigen	E7	1	11			Befehlszähler nach Stapel, Startadresse 0020
RST	5	code-eigen	EF	1	11			Befehlszähler nach Stapel, Startadresse 0028
RST	6	code-eigen	F7	1	11			Befehlszähler nach Stapel, Startadresse 0030
RST	7	code-eigen	FF	1	11			Befehlszähler nach Stapel, Startadresse 0038

RST bedeutet ReSTart = starte wieder.

c. Die RST-Befehle können auch vom Programm abgesetzt werden und starten ein Unterbrechungsprogramm, das auf festen Adressen in den unteren 64 Bytes des Speichers beginnt. Von diesen Stellen aus kann man beliebig in weitere Programmteile springen, falls die zur Verfügung stehenden acht Bytes bis zur nächsten Anfangsadresse nicht ausreichen. Der RST 0 Befehl wirkt wie ein RESET.

Der Rücksprung in das unterbrochene Programm erfolgt mit dem Rücksprungbefehl wie bei einem Unterprogramm, der die Rücksprungadresse wieder aus dem Stapel holt und in das Befehlszählregister lädt.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C		
RET		code-eigen	C9	1	10			Befehlszähler vom Stapel, Rücksprung

RET bedeutet RETurn = springe zurück.

Der Abschnitt 7.2.3. enthält die bedingten Rücksprungbefehle.

Der Interrupt wird nur ausgeführt, wenn das INTE-Flip-Flop 1 ist. Die beiden folgenden Befehle sperren den Interrupt bzw. geben ihn frei.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H-P-C		
EI		code-eigen	FB	1	4			Interrupt erlaubt, INTE-Flip-Flop ← 1
DI		code-eigen	F3	1	4			Interrupt gesperrt, INTE-Flip-Flop ← 0

EI bedeutet Enable Interrupt = erlaube Interrupt
DI bedeutet Disable Interrupt = sperre Interrupt

Der folgende Befehl bringt den Prozessor in den Wartezustand, aus dem er nur durch ein RESET oder einen Interrupt wieder gestartet werden kann.

Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
HLT		code-eigen	76	1	7			Anhalten bis Interrupt

HLT bedeutet HaLT

Ist der Interrupt gesperrt (INTE-Flip-Flop = 0), so wirkt nur noch ein RESET.

Bild 114 zeigt ein Beispiel für die Interrupt-Adressen des 8080.

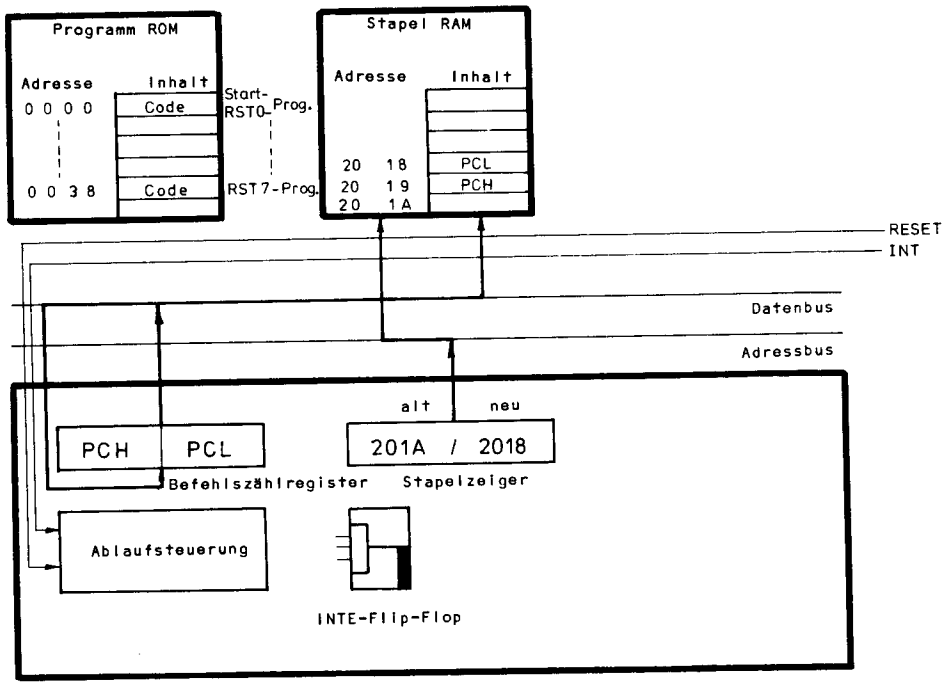


Bild 114: Interrupt des 8080

Die Speicherstelle 0000 enthält den ersten Befehl des Hauptprogramms, das bei RESET oder bei einem Interrupt mit dem Byte C7 auf dem Datenbus oder mit dem Befehl RST 0 gestartet wird. Die Bitkombination 11111111 = FF auf dem Datenbus nach einem Interrupt startet ein Programm mit der Startadresse 0038. Der Prozessor rettet bei allen Unterprogrammaufrufen und allen RST-Befehlen nur die Rücksprungadresse in den Stapel.

7.3.4 Die Interrupt-Möglichkeiten des 8085

Der Mikroprozessor Intel 8085 hat sechs Interrupt-Möglichkeiten.

a. Das RESET setzt wie beim 8080 den Befehlszähler auf 0000 und startet ein Programm mit der Startadresse 0000.
b. Der INTR-Interrupt arbeitet wie der INT-Interrupt des 8080 mit Funktionscodes (z.B. RST-Befehlen), die über den Datenbus geliefert werden. Für die acht verschiedenen RST-Befehle gibt es acht verschiedene Startadressen in den ersten 64 Bytes des Speicherbereiches.
c. Zusätzlich gibt es vier Interrupt-Eingänge, die mit vier festgelegten Startadressen arbeiten:

Name	Startadresse
TRAP	0024
RST5.5	002C
RST6.5	0034
RST7.5	003C

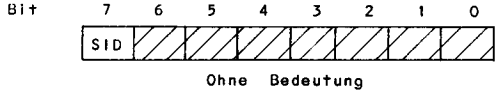
Da die Bereiche zwischen den RST-Startadressen liegen und nur vier Bytes umfassen, wird man von diesen Stellen aus in weitere Programmteile springen.
Der TRAP-Interrupt ist nicht sperrbar und wird vor allen anderen ausgeführt.
Die RST-Eingänge können über ein zusätzliches Interrupt-Register gesperrt und abgefragt werden.

Der Mikroprozessor 8085 hat einen Eingang und einen Ausgang für seriellen Datentransport. Die beiden folgenden Befehle lesen und schreiben sowohl das Interrupt-Register als auch die beiden Leitungen.

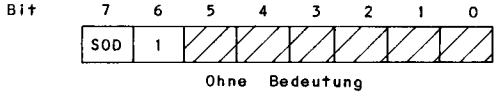
Befehl	Operand	Adressierung	Hexa	Bytes	Zyk.	Bedingungen NZ-H	-P-C	
RIM		code-eigen	20	1	4			Lade Akku mit Interrupt-Register und Eingabe
SIM		code-eigen	30	1	4			Speichere A nach Interrupt-Reg. oder Ausgabe

RIM bedeutet Read Interrupt Mask = lies Interrupt-Maske
SIM bedeutet Set Interrupt Mask = setze Interrupt-Maske

Für die serielle Ein/Ausgabe von Daten hat das Interrupt-Register folgende Bedeutung:
Beim Lesen des Registers mit dem Befehl RIM enthält Bit 7 das am SID-Anschluß anliegende Bit.

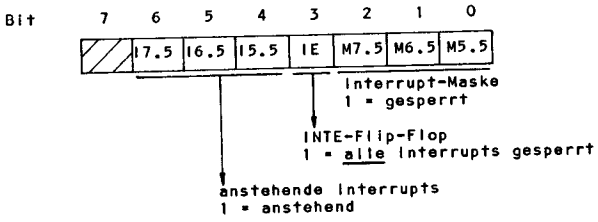


Zum Ausgeben eines Bits über den SOD-Anschluß mit dem Befehl SIM muß Bit 6 eine 1 enthalten (Einschalten des Ausgabe-Modus); Bit 7 enthält das auszugebende Bit.

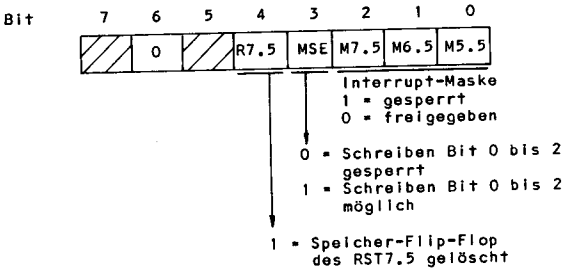


Für Interrupt-Behandlung hat das Interrupt-Register folgende Bedeutung:

Durch Lesen des Registers mit dem Befehl RIM können die 5.5-, 6.5- und 7.5-Interruptbedingungen kontrolliert werden.



Durch Schreiben des Registers mit dem Befehl SIM können die 5.5-, 6.5- und 7.5-Interruptbedingungen beeinflußt werden. Dazu muß Bit 3 auf 1 gesetzt sein.



Beispiel:

Das System ist für die Annahme des RST 7.5-Interrupt bereit zu machen und soll dann auf einen Interrupt warten.

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,18H
	18			
2882	30		SIM	
2883	FB		EI	
2884	76		HLT	
			END	

Das folgende Testprogramm gilt nur für den Bausatz SDK 85, bei dem die Taste VECT/INTR auf den RST 7.5-Interrupt arbeitet. Das Betriebsprogramm verlangt, daß der Benutzer in den Speicherstellen 20D4 bis 20D6 einen Sprungbefehl ablegt, der in das Unterbrechungsprogramm führt. Im Hauptprogramm muß der Benutzer vorher das I-Register auf 08 setzen und die Interruptsperre durch den Befehl EI aufheben.

Hauptprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	2880H
2880	3E	START	MVI	A,0FFH
	FF			
2882	D3		OUT	02H
	02			
2884	AF		XRA	A
2885	D3		OUT	03H
	03			
2887	3E		MVI	A,18H
	18			
2889	30		SIM	
288A	FB	BALL	EI	
288B	76		HLT	
288C	C3		JMP	BALL
	8A			
	28			

Unterbrechungsprogramm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
			ORG	20D4H
20D4	C3	ADR	JMP	INA
	90			
	28			
			ORG	2890H
2890	DB	INA	IN	01H
	01			
2892	D3		OUT	00H
	00			
2894	C9		RET	
			END	

Das Testprogramm wird bei der Adresse 2880 gestartet, programmiert die Ein/Ausgabe, hebt die Interrupt-Sperren auf und wartet dann auf einen Interrupt. Nach dem Drücken der Interrupt-Taste übernimmt das Unterbrechungsprogramm den an den Kippschaltern eingestellten Wert, zeigt ihn an den Leuchtdioden an und kehrt wieder in das Hauptprogramm zurück, das auf eine neue Unterbrechung wartet.

Übungen zum Abschnitt 7. A

1.Aufgabe

Ein Unterprogramm soll mit einer Grundzeit von 2ms verzögern, ein Verzögerungsfaktor wird vom Hauptprogramm in einem Register übergeben. Ein Test-Hauptprogramm holt sich diesen Faktor von den Kippschaltern nach Bild 48 und läßt die Bitkombination 00000001 verzögert auf den Ausgabe-Leuchtdioden rotieren.

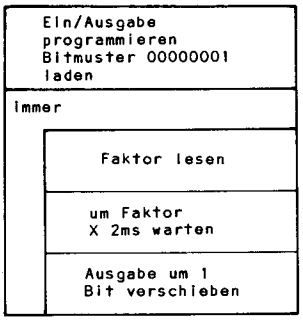
2.Aufgabe

An den Kippschaltern einer Schaltung nach Bild 48 ist eine zweiziffrige BCD-Zahl einzustellen, mit einem unterbrechungsfesten (re-entrant) Unterprogramm in eine Dualzahl umzuwandeln und an den Leuchtdioden anzuzeigen.Die Übernahme der BCD-Zahl und die Anzeige auf den Leuchtdioden soll durch einen Interrupt ausgelöst werden.

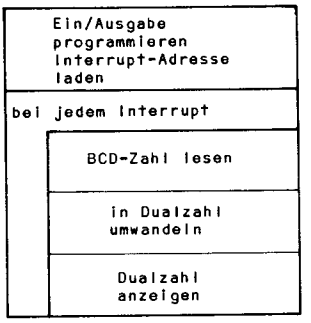
3.Aufgabe

An den Kippschaltern einer Schaltung nach Bild 48 sind Bitkombinationen einzustellen und an den Leuchtdioden anzuzeigen. Bei jedem Interrupt sind die Werte mit einer bestimmten Adresse beginnend nacheinander im Speicher abzulegen. Erscheint eine bestimmte Bitkombination (z.B. AA), so ist die Adresse des ersten eingegebenen Wertes in den Befehlszähler zu bringen.
Mit dieser Aufgabe ist es möglich, ein Programm einzugeben und zu starten. Als Endmarke wähle man eine Bitkombination, die in dem einzugebenden Programm nicht vorkommt. Da beim KIM die Leitung B6 fehlt, muß hier jedes Byte in zwei Teilen nur über die vier wertniedrigsten Kippschalter eingegeben werden.

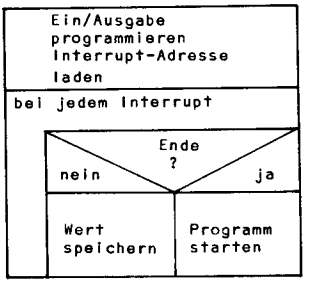
1.Aufgabe



2.Aufgabe



3.Aufgabe



8. Lösungen der Übungsaufgaben

Lösungen zum Abschnitt 1.1. und 1.2.

1.Aufgabe:Die Dezimalzahl 100 ist
1100100 dual
144 oktal
64 hexadezimal
1210 zur Basis 4

2.Aufgabe: Duales Rechnen
a. 00101111 + 00001011 = 00111010 = 58
b. 01100100 + 11011010 = 100111110 - 100000000= 00111110 = 62
c. 00100110 + 10011100 = 11000010 = -62
d. 11010001 + 11110101 = 111000110 - 100000000= 11000110 =-58

3.Aufgabe: Speicherinhalt
a. 136 als positive Dualzahl
b. -120 als vorzeichenbehaftete Dualzahl
c. 88 als BCD-Ziffern

4.Aufgabe: ICH HABS SATT

5.Aufgabe: Die Dezimalzahl 1000 ist
dual 1111101000
Basis 4 33220
oktal 1750
hexadezimal 3E8

Lösungen zum Abschnitt 1.3 und 1.4

1.Aufgabe:Wertetabelle eines Volladdierers

A	B	C	X	Y
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2.Aufgabe:
a. 0011 + 0101 + 0 = 1000 Z=0 ; N=1 ; V=1 ; C=0
b. 0011 - 0101 - 0 = 1110 Z=0 ; N=1 ; V=0 ; C=0
c. 0011 AND 0101 = 0001 Z=0 ; N=0 ; V=0 ; C=0
d. 0011 OR 0101 = 0111 Z=0 ; N=0 ; V=0 ; C=0
e. 0011 XOR 0101 = 0110 Z=0 ; N=0 ; V=0 ; C=0

3.Aufgabe: Addition
a. Ergebnis 0 1001 1000 hexadezimal 98
b. 152
c. -104
d. 98

Lösungen zum Abschnitt 1.5. und 1.6.

1.Aufgabe:Wertetabelle

S	R	Q	Q+	
0	0	0	0	speichern
0	0	1	1	
0	1	0	0	rücksetzen
0	1	1	0	
1	0	0	1	setzen
1	0	1	1	
1	1	0	0	verboten
1	1	1	0	

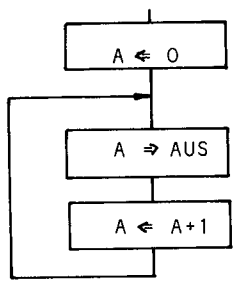
2.Aufgabe: Verschieben des Bitmusters 1011
a. logisch rechts 0101
b. logisch links 0110
c. zyklisch rechts 1101
d. zyklisch links 0111
e. algebraisch rechts 1101

3.Aufgabe:
Mit der Adresse 011 wird der Inhalt 0110 angesprochen.

Lösungen zum Abschnitt 1.7 bis 1.9.

1.Aufgabe:

Programmablaufplan



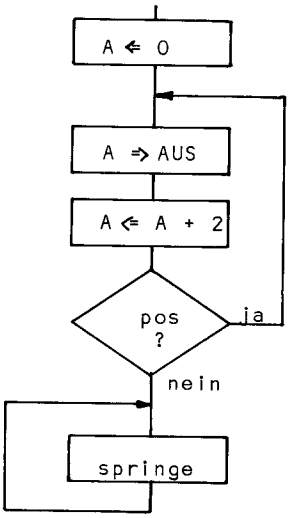
Programm

Name	Befehl	Operand
EINS	=	1
START	LDK	0
SUSI	SPE	AUS
	ADD	EINS
	SPR	SUSI

2.Aufgabe: PAUL nimmt den Inhalt 1001 an.
0001 UND 1111 gibt 0001
0001 PLUS 1000 gibt 1001

3. Aufgabe:

Programmablaufplan



Programm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
03	2	ZWEI	=	2
80	1	START	LDK	0
81	0			
82	2	MAXI	SPE	AUS
83	0			
84	1			
85	3		ADD	ZWEI
86	0			
87	3			
88	B		SPP	MAXI
89	8			
8A	2			
8B	A	SUSI	SPR	SUSI
8C	8			
8D	B			

Lösungen zum Abschnitt 3.1. bis 3.5.

Alle drei Aufgaben wurden zu einem Testprogramm zusammengefaßt. Am Ende des Programms steht eine Warteschleife.

KIM

```
                        *=$0000
0000            RA      *=*+1
0001            RX      *=*+1
0002            RY      *=*+1
0003            RS      *=*+1

                        *=$0300
0300 8D 00 AUF1 STA     RA
0303 8E 01 00   STX     RX
0306 8C 02 00   STY     RY
0309 BA         TSX
030A 8E 03 00   STX     RS
030D A9 00 AUF2 LDA     #$00
030F AA         TAX
0310 A8         TAY
0311 AE 03 00 AUF3 LDX  RS
0314 9A         TXS
0315 AC 02 00   LDY     RY
0318 AE 01 00   LDX     RX
```

6800

```
                        ORG     $0000
0000            RA      RMB     1
0001            RB      RMB     1
0002            RBE     RMB     1
0003            RX      RMB     2
0005            RS      RMB     2

                        ORG     $0180
0180 B7 00 00 AUF1 STA A RA
0183 F7 00 01   STA B   RB
0186 07         TPA
0187 B7 00 02   STA A   RBE
018A FF 00 03   STX     RX
018D BF 00 05   STS     RS
0190 4F AUF2    CLR A
0191 5F         CLR B
0192 06         TAP
0193 CE 00 00   LDX     #$0000
0196 BE 00 05 AUF3 LDS  RS
0199 FE         LDX     RX
```

8080/8085

```
                        ORG     2000H
2000            RA      DS      1
2001            RHL     DS      2
2003            RBC     DS      2
2005            RDE     DS      2

                        ORG     2880H
2880 32 00 20 AUF1 STA  RA
2883 22 01 20   SHLD    RHL
2886 EB         XCHG
2887 22 05 20   SHLD    RDE
288A 60         MOV     H,B
288B 69         MOV     L,C
288C 22 03 20   SHLD    RBC
288F 3E 00 AUF2 MVI     A,00H
2891 47         MOV     B,A
2892 4F         MOV     C,A
2893 57         MOV     D,A
2894 5F         MOV     E,A
2895 67         MOV     H,A
2896 6F         MOV     L,A
2897 2A 03 20 AUF3 LHLD RBC
289A 44         MOV     B,H
```

```
031B  AD          LDA    RA
      00
      00
031E  4C   OTTO   JMP    OTTO
      1E
      03
                  .END
```

```
      00
      03
019C  B6          LDA A  RBE
      00
      02
019F  06          TAP
01A0  F6          LDA B  RB
      00
      01
01A3  B6          LDA A  RA
      00
      00
01A6  7E   OTTO   JMP    OTTO
      01
      A6
                  END
```

A → Bad. Mag.

```
289B  4D          MOV    C,L
289C  2A          LHLD   RDE
      05
      20
289F  EB          XCHG
28A0  2A          LHLD   RHL
      01
      20
28A3  3A          LDA    RA
      00
      20
28A6  C3   OTTO   JMP    OTTO
      A6
      28
                  END
```

Lösungen zum Abschnitt 3.6.

Für die Adressen der Ein/Ausgaberegister werden hexadezimale Bezeichnungen verwendet.

1.Aufgabe:

KIM

```
                  *=$0000
0000        AEIN  *=*+1
0001        BEIN  *=*+1

                  *=$0300
0300  A9   START  LDA    #$00
      00
0302  8D          STA    $1701
      01
      17
0305  8D          STA    $1703
      03
      17
0308  AD          LDA    $1700
      00
      17
030B  8D          STA    AEIN
      00
      00
030E  AD          LDA    $1702
      02
      17
0311  8D          STA    BEIN
      01
      00
0314  4C   OTTO   JMP    OTTO
      14
      03
                  .END
```

6800

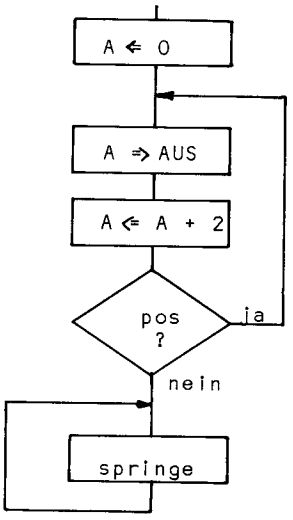
```
                  ORG    $0000
0000        AEIN  RMB    1
0001        BEIN  RMB    1

                  ORG    $0180
0180  86   START  LDA A  #$04
      04
0182  B7          STA A  $8005
      80
      05
0185  B7          STA A  $8007
      80
      07
0188  B6          LDA A  $8004
      80
      04
018B  B7          STA A  AEIN
      00
      00
018E  B6          LDA A  $8006
      80
      06
0191  B7          STA A  BEIN
      00
      01
0194  7E   OTTO   JMP    OTTO
      01
      94
                  END
```

8080/8085

```
                  ORG    2000H
2000        AEIN  DS     1
2001        BEIN  DS     1

                  ORG    2880H
2880  3E   START  MVI    A,00H
      00
2882  D3          OUT    02H
      02
2884  D3          OUT    03H
      03
2886  DB          IN     00H
      00
2888  32          STA    AEIN
      00
      20
288B  DB          IN     01H
      01
288D  32          STA    BEIN
      01
      20
2890  C3   OTTO   JMP    OTTO
      90
      28
                  END
```

3. Aufgabe:

Programmablaufplan



Programm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
03	2	ZWEI	=	2
80	1	START	LDK	0
81	0			
82	2	MAXI	SPE	AUS
83	0			
84	1			
85	3		ADD	ZWEI
86	0			
87	3			
88	B		SPP	MAXI
89	8			
8A	2			
8B	A	SUSI	SPR	SUSI
8C	8			
8D	B			

Lösungen zum Abschnitt 3.1. bis 3.5.

Alle drei Aufgaben wurden zu einem Testprogramm zusammengefaßt. Am Ende des Programms steht eine Warteschleife.

KIM

```
                        *=$0000
0000          RA        *=*+1
0001          RX        *=*+1
0002          RY        *=*+1
0003          RS        *=*+1

                        *=$0300
0300  8D      AUF1      STA     RA
      00
      00
0303  8E                STX     RX
      01
      00
0306  8C                STY     RY
      02
      00
0309  BA                TSX
030A  8E                STX     RS
      03
      00
030D  A9      AUF2      LDA     #$00
      00
030F  AA                TAX
0310  A8                TAY
0311  AE      AUF3      LDX     RS
      03
      00
0314  9A                TXS
0315  AC                LDY     RY
      02
      00
0318  AE                LDX     RX
      01
      00
```

6800

```
                        ORG     $0000
0000          RA        RMB     1
0001          RB        RMB     1
0002          RBE       RMB     1
0003          RX        RMB     2
0005          RS        RMB     2

                        ORG     $0180
0180  B7      AUF1      STA A   RA
      00
      00
0183  F7                STA B   RB
      00
      01
0186  07                TPA
0187  B7                STA A   RBE
      00
      02
018A  FF                STX     RX
      00
      03
018D  BF                STS     RS
      00
      05
0190  4F      AUF2      CLR A
0191  5F                CLR B
0192  06                TAP
0193  CE                LDX     #$0000
      00
      00
0196  BE      AUF3      LDS     RS
      00
      05
0199  FE                LDX     RX
```

8080/8085

```
                        ORG     2000H
2000          RA        DS      1
2001          RHL       DS      2
2003          RBC       DS      2
2005          RDE       DS      2

                        ORG     2880H
2880  32      AUF1      STA     RA
      00
      20
2883  22                SHLD    RHL
      01
      20
2886  EB                XCHG
2887  22                SHLD    RDE
      05
      20
288A  60                MOV     H,B
288B  69                MOV     L,C
288C  22                SHLD    RBC
      03
      20
288F  3E      AUF2      MVI     A,00H
      00
2891  47                MOV     B,A
2892  4F                MOV     C,A
2893  57                MOV     D,A
2894  5F                MOV     E,A
2895  67                MOV     H,A
2896  6F                MOV     L,A
2897  2A      AUF3      LHLD    RBC
      03
      20
289A  44                MOV     B,H
```

```
031B AD 00 00        LDA     RA
031E 4C 1E 03  OTTO  JMP     OTTO
                     .END
```

```
        00 03
019C B6 00 02        LDA A   RBE
019F 06              TAP
01A0 F6 00 01        LDA B   RB
01A3 B6 00 00        LDA A   RA
01A6 7E 01 A6  OTTO  JMP     OTTO
                     END
```

```
289B 4D              MOV     C,L
289C 2A 05 20        LHLD    RDE
289F EB              XCHG
28A0 2A 01 20        LHLD    RHL
28A3 3A 00 20        LDA     RA
28A6 C3 A6 28  OTTO  JMP     OTTO
                     END
```

Lösungen zum Abschnitt 3.6.

Für die Adressen der Ein/Ausgaberegister werden hexadezimale Bezeichnungen verwendet.

1.Aufgabe:

KIM

```
                     *=$0000
0000         AEIN    *=*+1
0001         BEIN    *=*+1

                     *=$0300
0300 A9 00   START   LDA     #$00
0302 8D 01 17        STA     $1701
0305 8D 03 17        STA     $1703
0308 AD 00 17        LDA     $1700
030B 8D 00 00        STA     AEIN
030E AD 02 17        LDA     $1702
0311 8D 01 00        STA     BEIN
0314 4C 14 03  OTTO  JMP     OTTO
                     .END
```

6800

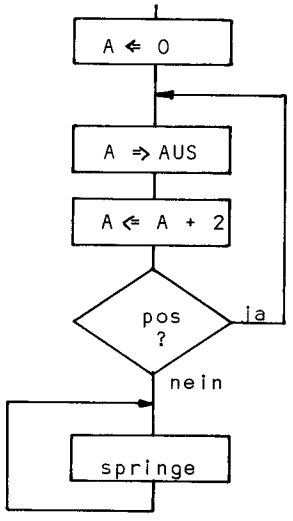
```
                     ORG     $0000
0000         AEIN    RMB     1
0001         BEIN    RMB     1

                     ORG     $0180
0180 B6 04   START   LDA A   #$04
0182 B7 80 05        STA A   $8005
0185 B7 80 07        STA A   $8007
0188 B6 80 04        LDA A   $8004
018B B7 00 00        STA A   AEIN
018E B6 80 06        LDA A   $8006
0191 B7 00 01        STA A   BEIN
0194 7E 01 94  OTTO  JMP     OTTO
                     END
```

8080/8085

```
                     ORG     2000H
2000         AEIN    DS      1
2001         BEIN    DS      1

                     ORG     2880H
2880 3E 00   START   MVI     A,00H
2882 D3 02           OUT     02H
2884 D3 03           OUT     03H
2886 DB 00           IN      00H
2888 32 00 20        STA     AEIN
288B DB 01           IN      01H
288D 32 01 20        STA     BEIN
2890 C3 90 28  OTTO  JMP     OTTO
                     END
```

3. Aufgabe:

Programmablaufplan



Programm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
03	2	ZWEI	=	2
80	1	START	LDK	0
81	0			
82	2	MAXI	SPE	AUS
83	0			
84	1			
85	3		ADD	ZWEI
86	0			
87	3			
88	B		SPP	MAXI
89	8			
8A	2			
8B	A	SUSI	SPR	SUSI
8C	8			
8D	B			

Lösungen zum Abschnitt 3.1. bis 3.5.

Alle drei Aufgaben wurden zu einem Testprogramm zusammengefaßt. Am Ende des Programms steht eine Warteschleife.

KIM

```
                    *=$0000
0000        RA      *=*+1
0001        RX      *=*+1
0002        RY      *=*+1
0003        RS      *=*+1

                    *=$0300
0300  8D    AUF1   STA     RA
      00
      00
0303  8E           STX     RX
      01
      00
0306  8C           STY     RY
      02
      00
0309  BA           TSX
030A  8E           STX     RS
      03
      00
030D  A9    AUF2   LDA     #$00
      00
030F  AA           TAX
0310  A8           TAY
0311  AE    AUF3   LDX     RS
      03
      00
0314  9A           TXS
0315  AC           LDY     RY
      02
      00
0318  AE           LDX     RX
      01
      00
```

6800

```
                    ORG     $0000
0000        RA      RMB     1
0001        RB      RMB     1
0002        RBE     RMB     1
0003        RX      RMB     2
0005        RS      RMB     2

                    ORG     $0180
0180  B7    AUF1   STA A   RA
      00
0183  F7           STA B   RB
      00
      01
0186  07           TPA
0187  B7           STA A   RBE
      00
      02
018A  FF           STX     RX
      00
      03
018D  BF           STS     RS
      00
      05
0190  4F    AUF2   CLR A
0191  5F           CLR B
0192  06           TAP
0193  CE           LDX     #$0000
      00
      00
0196  BE    AUF3   LDS     RS
      00
      05
0199  FE           LDX     RX
```

8080/8085

```
                    ORG     2000H
2000        RA      DS      1
2001        RHL     DS      2
2003        RBC     DS      2
2005        RDE     DS      2

                    ORG     2880H
2880  32    AUF1   STA     RA
      00
      20
2883  22           SHLD    RHL
      01
      20
2886  EB           XCHG
2887  22           SHLD    RDE
      05
      20
288A  60           MOV     H,B
288B  69           MOV     L,C
288C  22           SHLD    RBC
      03
      20
288F  3E    AUF2   MVI     A,00H
      00
2891  47           MOV     B,A
2892  4F           MOV     C,A
2893  57           MOV     D,A
2894  5F           MOV     E,A
2895  67           MOV     H,A
2896  6F           MOV     L,A
2897  2A    AUF3   LHLD    RBC
      03
      20
289A  44           MOV     B,H
```

```
031B AD       LDA   RA
     00
     00
031E 4C 1E 03 OTTO  JMP   OTTO
                    .END
```

```
     00
     03
019C B6       LDA A RBE
     00
     02
019F 06       TAP
01A0 F6       LDA B RB
     00
     01
01A3 B6       LDA A RA
     00
     00
01A6 7E 01 A6 OTTO  JMP   OTTO
                    END
```

```
289B 4D       MOV   C,L
289C 2A 05 20 LHLD  RDE
289F EB       XCHG
28A0 2A 01 20 LHLD  RHL
28A3 3A 00 20 LDA   RA
28A6 C3 A6 28 OTTO  JMP   OTTO
                    END
```

Lösungen zum Abschnitt 3.6.

Für die Adressen der Ein/Ausgaberegister werden hexadezimale Bezeichnungen verwendet.

1.Aufgabe:

KIM

```
                    *=$0000
0000        AEIN    *=*+1
0001        BEIN    *=*+1

                    *=$0300
0300 A9 00  START   LDA   #$00
0302 8D 01 17       STA   $1701
0305 8D 03 17       STA   $1703
0308 AD 00 17       LDA   $1700
030B 8D 00 00       STA   AEIN
030E AD 02 17       LDA   $1702
0311 8D 01 00       STA   BEIN
0314 4C 14 03 OTTO  JMP   OTTO
                    .END
```

6800

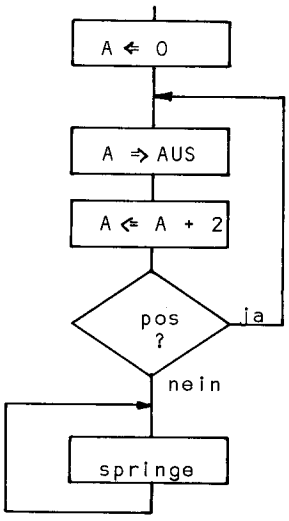
```
                    ORG   $0000
0000        AEIN    RMB   1
0001        BEIN    RMB   1

                    ORG   $0180
0180 86 04  START   LDA A #$04
0182 B7 80 05       STA A $8005
0185 B7 80 07       STA A $8007
0188 B6 80 04       LDA A $8004
018B B7 00 00       STA A AEIN
018E B6 80 06       LDA A $8006
0191 B7 00 01       STA A BEIN
0194 7E 01 94 OTTO  JMP   OTTO
                    END
```

8080/8085

```
                    ORG   2000H
2000        AEIN    DS    1
2001        BEIN    DS    1

                    ORG   2880H
2880 3E 00  START   MVI   A,00H
2882 D3 02          OUT   02H
2884 D3 03          OUT   03H
2886 DB 00          IN    00H
2888 32 00 20       STA   AEIN
288B DB 01          IN    01H
288D 32 01 20       STA   BEIN
2890 C3 90 28 OTTO  JMP   OTTO
                    END
```

3. Aufgabe:

Programmablaufplan



Programm

Hexadezimal Adresse	Inhalt	Symbolisch Name	Befehl	Operand
03	2	ZWEI	=	2
80	1	START	LDK	0
81	0			
82	2	MAXI	SPE	AUS
83	0			
84	1			
85	3		ADD	ZWEI
86	0			
87	3			
88	B		SPP	MAXI
89	8			
8A	2			
8B	A	SUSI	SPR	SUSI
8C	8			
8D	B			

Lösungen zum Abschnitt 3.1. bis 3.5.

Alle drei Aufgaben wurden zu einem Testprogramm zusammengefaßt. Am Ende des Programms steht eine Warteschleife.

KIM

```
                        *=$0000
0000            RA      *=*+1
0001            RX      *=*+1
0002            RY      *=*+1
0003            RS      *=*+1

                        *=$0300
0300  8D 00 00  AUF1    STA     RA
0303  8E 01 00          STX     RX
0306  8C 02 00          STY     RY
0309  BA                TSX
030A  8E 03 00          STX     RS
030D  A9 00     AUF2    LDA     #$00
030F  AA                TAX
0310  A8                TAY
0311  AE 03 00  AUF3    LDX     RS
0314  9A                TXS
0315  AC 02 00          LDY     RY
0318  AE 01 00          LDX     RX
```

6800

```
                        ORG     $0000
0000            RA      RMB     1
0001            RB      RMB     1
0002            RBE     RMB     1
0003            RX      RMB     2
0005            RS      RMB     2

                        ORG     $0180
0180  B7 00 00  AUF1    STA A   RA
0183  F7 00 01          STA B   RB
0186  07                TPA
0187  B7 00 02          STA A   RBE
018A  FF 00 03          STX     RX
018D  BF 00 05          STS     RS
0190  4F        AUF2    CLR A
0191  5F                CLR B
0192  06                TAP
0193  CE 00 00          LDX     #$0000
0196  BE 00 05  AUF3    LDS     RS
0199  FE                LDX     RX
```

8080/8085

```
                        ORG     2000H
2000            RA      DS      1
2001            RHL     DS      2
2003            RBC     DS      2
2005            RDE     DS      2

                        ORG     2880H
2880  32 00 20  AUF1    STA     RA
2883  22 01 20          SHLD    RHL
2886  EB                XCHG
2887  22 05 20          SHLD    RDE
288A  60                MOV     H,B
288B  69                MOV     L,C
288C  22 03 20          SHLD    RBC
288F  3E 00     AUF2    MVI     A,00H
2891  47                MOV     B,A
2892  4F                MOV     C,A
2893  57                MOV     D,A
2894  5F                MOV     E,A
2895  67                MOV     H,A
2896  6F                MOV     L,A
2897  2A 03 20  AUF3    LHLD    RBC
289A  44                MOV     B,H
```

```
031B  AD        LDA   RA
      00
      00
031E  4C  OTTO  JMP   OTTO
      1E
      03
                .END
```

```
      00
      03
019C  B6        LDA A RBE
      00
      02
019F  06        TAP
01A0  F6        LDA B RB
      00
      01
01A3  B6        LDA A RA
      00
      00
01A6  7E  OTTO  JMP   OTTO
      01
      A6

                END
```

```
289B  4D        MOV   C,L
289C  2A        LHLD  RDE
      05
      20
289F  EB        XCHG
28A0  2A        LHLD  RHL
      01
      20
28A3  3A        LDA   RA
      00
      20
28A6  C3  OTTO  JMP   OTTO
      A6
      28
                END
```

Lösungen zum Abschnitt 3.6.

Für die Adressen der Ein/Ausgaberegister werden hexadezimale Bezeichnungen verwendet.

1.Aufgabe:

KIM

```
                *=$0000
0000      AEIN  *=*+1
0001      BEIN  *=*+1

                *=$0300
0300  A9  START LDA   #$00
      00
0302  8D        STA   $1701
      01
      17
0305  8D        STA   $1703
      03
      17
0308  AD        LDA   $1700
      00
      17
030B  8D        STA   AEIN
      00
      00
030E  AD        LDA   $1702
      02
      17
0311  8D        STA   BEIN
      01
      00
0314  4C  OTTO  JMP   OTTO
      14
      03
                .END
```

6800

```
                ORG   $0000
0000      AEIN  RMB   1
0001      BEIN  RMB   1

                ORG   $0180
0180  86  START LDA A #$04
      04
0182  B7        STA A $8005
      80
      05
0185  B7        STA A $8007
      80
      07
0188  B6        LDA A $8004
      80
      04
018B  B7        STA A AEIN
      00
      00
018E  B6        LDA A $8006
      80
      06
0191  B7        STA A BEIN
      00
      01
0194  7E  OTTO  JMP   OTTO
      01
      94
                END
```

8080/8085

```
                ORG   2000H
2000      AEIN  DS    1
2001      BEIN  DS    1

                ORG   2880H
2880  3E  START MVI   A,00H
      00
2882  D3        OUT   02H
      02
2884  D3        OUT   03H
      03
2886  DB        IN    00H
      00
2888  32        STA   AEIN
      00
      20
288B  DB        IN    01H
      01
288D  32        STA   BEIN
      01
      20
2890  C3  OTTO  JMP   OTTO
      90
      28
                END
```

2.Aufgabe:

KIM

```
                *=$0300
0300 A9 FF START LDA  #$FF
0302 8D 01 17    STA  $1701
0305 8D 03 17    STA  $1703
0308 8D 00 17    STA  $1700
030B 8D 02 17    STA  $1702
030E 4C 0E 03 OTTO JMP  OTTO
                 .END
```

6800

```
0180 73 80 00 START ORG  $0180
                    COM  $8004
0183 73 80 04       COM  $8006
0186 86 04          LDA A #$04
0188 B7 80 05       STA A $8005
018B B7 80 07       STA A $8007
018E 86 FF          LDA A #$FF
0190 B7 80 04       STA A $8004
0193 B7 80 06       STA A $8006
0196 7E 01 96 BOY   JMP  BOY
                    END
```

8080/8085

```
                ORG  2880H
2880 3E FF START MVI  A,0FFH
2882 D3 02       OUT  02H
2884 D3 03       OUT  03H
2886 D3 00       OUT  00H
2888 D3 01       OUT  01H
288A C3 8A 28 GIRL JMP  GIRL
                 END
```

3.Aufgabe:

KIM

```
                *=$0300
0300 A9 00 START LDA  #$00
0302 8D 03 17    STA  $1703
0305 A9 FF       LDA  #$FF
0307 8D 01 17    STA  $1701
030A AD 02 17 MAXI LDA  $1702
030D 8D 00 17    STA  $1700
0310 4C 0A 03    JMP  MAXI
                 .END
```

6800

```
0180 73 80 04 START ORG  $0180
                    COM  $8004
0183 86 04          LDA A #$04
0185 B7 80 05       STA A $8005
0188 B7 80 07       STA A $8007
018B B6 80 06 PAUL  LDA A $8006
018E B7 80 04       STA A $8004
0191 7E 01 8B       JMP  PAUL
                    END
```

8080/8085

```
                ORG  2880H
2880 3E FF START MVI  A,0FFH
2882 D3 02       OUT  02H
2884 3E 00       MVI  A,00H
2886 D3 03       OUT  03H
2888 DB 01 MAXI  IN   01H
288A D3 00       OUT  00H
288C C3 88 28    JMP  MAXI
                 END
```

Lösungen zum Abschnitt 4.2. und 4.3.

1.Aufgabe:

KIM

```
                    *=$0300
0300 A9 FF  START   LDA   #$FF
0302 8D 01 17       STA   $1701
0305 A9 00          LDA   #$00
0307 8D 03 17       STA   $1703
030A AD 02 17 MIMI  LDA   $1702
030D D0 02          BNE   RADI
030F A9 FF          LDA   #$FF
0311 8D 00 17 RADI  STA   $1700
0314 4C 0A 03       JMP   MIMI
                    .END
```

6800

```
                    ORG   $0180
0180 73 80  START   COM   $8004
0183 86 04          LDA A #$04
0185 B7 80 05       STA A $8005
0188 B7 80 07       STA A $8007
018B B6 80 06 HUGO  LDA A $8006
018E 26 02          BNE   SUSI
0190 86 FF          LDA A #$FF
0192 B7 80 04 SUSI  STA A $8004
0195 7E 01 8B       JMP   HUGO
                    END
```

8080/8085

```
                    ORG   2880H
2880 3E FF  START   MVI   A,0FFH
2882 D3 02          OUT   02H
2884 3E 00          MVI   A,00H
2886 D3 03          OUT   03H
2888 DB 01  SUSI    IN    01H
288A C6 00          ADI   00H
288C C2 91 28       JNZ   MAX
288F 3E FF          MVI   A,0FFH
2891 D3 00  MAX     OUT   00H
2893 C3 88 28       JMP   SUSI
                    END
```

2.Aufgabe:

KIM

```
                    *=$0300
0300 A9 FF  START   LDA   #$FF
0302 8D 01 17       STA   $1701
0305 A9 00          LDA   #$00
0307 8D 03 17       STA   $1703
030A AD 02 17 MINI  LDA   $1702
030D D0 06          BNE   MAXI
030F 8D 00 17       STA   $1700
0312 4C 12 03 OTTO  JMP   OTTO
0315 10 02  MAXI    BPL   MONI
0317 A9 FF          LDA   #$FF
0319 8D 00 17 MONI  STA   $1700
031C 4C 0A 03       JMP   MINI
                    .END
```

6800

```
                    ORG   $0180
0180 73 80  START   COM   $8004
0183 86 04          LDA A #$04
0185 B7 80 05       STA A $8005
0188 B7 80 07       STA A $8007
018B B6 80 06 WILLI LDA A $8006
018E 26 05          BNE   HUGO
0190 B7 80 04       STA A $8004
0193 20 FE  OTTO    BRA   OTTO
0195 2A 02  HUGO    BPL   ADAM
0197 86 FF          LDA A #$FF
0199 B7 80 04 ADAM  STA A $8004
019C 7E 01 8B       JMP   WILLI
                    END
```

8080/8085

```
                    ORG   2880H
2880 3E FF  START   MVI   A,0FFH
2882 D3 02          OUT   02H
2884 3E 00          MVI   A,00H
2886 D3 03          OUT   03H
2888 DB 01  SUSI    IN    01H
288A C6 00          ADI   00H
288C C2 94 28       JNZ   PAUL
288F D3 00          OUT   00H
2891 C3 91 28 OTTO  JMP   OTTO
2894 F2 99 28 PAUL  JP    ADAM
2897 3E FF          MVI   A,0FFH
2899 D3 00  ADAM    OUT   00H
289B C3 88 28       JMP   SUSI
                    END
```

2.Aufgabe:

KIM

```
                 *=$0300
0300  A9  START  LDA    #$FF
      FF
0302  8D         STA    $1701
      01
      17
0305  8D         STA    $1703
      03
      17
0308  8D         STA    $1700
      00
      17
030B  8D         STA    $1702
      02
      17
030E  4C  OTTO   JMP    OTTO
      0E
      03
                 .END
```

6800

```
                 ORG    $0180
0180  73  START  COM    $8004
      80
      04
0183  73         COM    $8006
      80
      06
0186  86         LDA A  #$04
      04
0188  B7         STA A  $8005
      80
      05
018B  B7         STA A  $8007
      80
      07
018E  86         LDA A  #$FF
      FF
0190  B7         STA A  $8004
      80
      04
0193  B7         STA A  $8006
      80
      06
0196  7E  BOY    JMP    BOY
      01
      96
                 END
```

8080/8085

```
                 ORG    2880H
2880  3E  START  MVI    A,0FFH
      FF
2882  D3         OUT    02H
      02
2884  D3         OUT    03H
      03
2886  D3         OUT    00H
      00
2888  D3         OUT    01H
      01
288A  C3  GIRL   JMP    GIRL
      8A
      28
                 END
```

3.Aufgabe:

KIM

```
                 *=$0300
0300  A9  START  LDA    #$00
      00
0302  8D         STA    $1703
      03
      17
0305  A9         LDA    #$FF
      FF
0307  8D         STA    $1701
      01
      17
030A  AD  MAXI   LDA    $1702
      02
      17
030D  8D         STA    $1700
      00
      17
0310  4C         JMP    MAXI
      0A
      03
                 .END
```

6800

```
                 ORG    $0180
0180  73  START  COM    $8004
      80
      04
0183  86         LDA A  #$04
      04
0185  B7         STA A  $8005
      80
      05
0188  B7         STA A  $8007
      80
      07
018B  B6  PAUL   LDA A  $8006
      80
      06
018E  B7         STA A  $8004
      80
      04
0191  7E         JMP    PAUL
      01
      8B
                 END
```

8080/8085

```
                 ORG    2880H
2880  3E  START  MVI    A,0FFH
      FF
2882  D3         OUT    02H
      02
2884  3E         MVI    A,00H
      00
2886  D3         OUT    03H
      03
2888  DB  MAXI   IN     01H
      01
288A  D3         OUT    00H
      00
288C  C3         JMP    MAXI
      88
      28
                 END
```

Lösungen zum Abschnitt 4.2. und 4.3.

1.Aufgabe:

KIM

```
                    *=$0300
0300 A9      START  LDA    #$FF
     FF
0302 8D             STA    $1701
     01
     17
0305 A9             LDA    #$00
     00
0307 8D             STA    $1703
     03
     17
030A AD      MIMI   LDA    $1702
     02
     17
030D D0             BNE    RADI
     02
030F A9             LDA    #$FF
     FF
0311 8D      RADI   STA    $1700
     00
     17
0314 4C             JMP    MIMI
     0A
     03
                    .END
```

6800

```
                    ORG    $0180
0180 73      START  COM    $8004
     80
     04
0183 86             LDA A  #$04
     04
0185 B7             STA A  $8005
     80
     05
0188 B7             STA A  $8007
     80
     07
018B B6      HUGO   LDA A  $8006
     80
     06
018E 26             BNE    SUSI
     02
0190 86             LDA A  #$FF
     FF
0192 B7      SUSI   STA A  $8004
     80
     04
0195 7E             JMP    HUGO
     01
     8B
                    END
```

8080/8085

```
                    ORG    2880H
2880 3E      START  MVI    A,0FFH
     FF
2882 D3             OUT    02H
     02
2884 3E             MVI    A,00H
     00
2886 D3             OUT    03H
     03
2888 DB      SUSI   IN     01H
     01
288A C6             ADI    00H
     00
288C C2             JNZ    MAX
     91
     28
288F 3E             MVI    A,0FFH
     FF
2891 D3      MAX    OUT    00H
     00
2893 C3             JMP    SUSI
     88
     28
                    END
```

2.Aufgabe:

KIM

```
                    *=$0300
0300 A9      START  LDA    #$FF
     FF
0302 8D             STA    $1701
     01
     17
0305 A9             LDA    #$00
     00
0307 8D             STA    $1703
     03
     17
030A AD      MINI   LDA    $1702
     02
     17
030D D0             BNE    MAXI
     06
030F 8D             STA    $1700
     00
     17
0312 4C      OTTO   JMP    OTTO
     12
     03
0315 10      MAXI   BPL    MONI
     02
0317 A9             LDA    #$FF
     FF
0319 8D      MONI   STA    $1700
     00
     17
031C 4C             JMP    MINI
     0A
     03
                    .END
```

6800

```
                    ORG    $0180
0180 73      START  COM    $8004
     80
     04
0183 86             LDA A  #$04
     04
0185 B7             STA A  $8005
     80
     05
0188 B7             STA A  $8007
     80
     07
018B B6      WILLI  LDA A  $8006
     80
     06
018E 26             BNE    HUGO
     05
0190 B7             STA A  $8004
     80
     04
0193 20      OTTO   BRA    OTTO
     FE
0195 2A      HUGO   BPL    ADAM
     02
0197 86             LDA A  #$FF
     FF
0199 B7      ADAM   STA A  $8004
     80
     04
019C 7E             JMP    WILLI
     01
     8B
                    END
```

8080/8085

```
                    ORG    2880H
2880 3E      START  MVI    A,0FFH
     FF
2882 D3             OUT    02H
     02
2884 3E             MVI    A,00H
     00
2886 D3             OUT    03H
     03
2888 DB      SUSI   IN     01H
     01
288A C6             ADI    00H
     00
288C C2             JNZ    PAUL
     94
     28
288F D3             OUT    00H
     00
2891 C3      OTTO   JMP    OTTO
     91
     28
2894 F2      PAUL   JP     ADAM
     99
     28
2897 3E             MVI    A,0FFH
     FF
2899 D3      ADAM   OUT    00H
     00
289B C3             JMP    SUSI
     88
     28
                    END
```

3.Aufgabe:

KIM

```
                    *=$0300
0300 A9 FF  START   LDA     #$FF
0302 8D 01 17       STA     $1701
0305 A9 00          LDA     #$00
0307 8D 03 17       STA     $1703
030A AD 02 17 OTTO  LDA     $1702
030D F0 12          BEQ     KARL
030F 10 08          BPL     PAUL
0311 A9 66          LDA     #$66
0313 8D 00 17       STA     $1700
0316 4C 0A 03       JMP     OTTO
0319 A9 99  PAUL    LDA     #$99
031B 8D 00 17       STA     $1700
031E 4C 0A 03       JMP     OTTO
0321 A9 FF  KARL    LDA     #$FF
0323 8D 00 17       STA     $1700
0326 4C 0A 03       JMP     OTTO
                    .END
```

6800

```
                    ORG     $0180
0180 73 80  START   COM     $8004
0183 86 04          LDA A   #$04
0185 B7 80 05       STA A   $8005
0188 B7 80 07       STA A   $8007
018B 86 80  ADAM    LDA A   $8006
018E 27 06          BEQ     CESAR
0190 2A 12          BPL     BERTA
0192 86 66          LDA A   #$66
0194 B7 80 04       STA A   $8004
0197 7E 01 8B       JMP     ADAM
019A 86 99  BERTA   LDA A   #$99
019C B7 80 04       STA A   $8004
019F 7E 01 8B       JMP     ADAM
01A2 86 FF  CESAR   LDA A   #$FF
01A4 B7 80 04       STA A   $8004
01A7 7E 01 8B       JMP     ADAM
                    END
```

8080/8085

```
                    ORG     2880H
2880 3E FF  START   MVI     A,0FFH
2882 D3 02          OUT     02H
2884 3E 00          MVI     A,00H
2886 D3 03          OUT     03H
2888 DB 01  SUSI    IN      01H
288A C6 00          ADI     00H
288C CA A0 28       JZ      OTTO
288F F2 99 28       JP      MAXI
2892 3E 66          MVI     A,66H
2894 D3 00          OUT     00H
2896 C3 88 28       JMP     SUSI
2899 3E 99  MAXI    MVI     A,99H
289B D3 00          OUT     00H
289D C3 88 28       JMP     SUSI
28A0 3E FF  OTTO    MVI     A,0FFH
28A2 D3 00          OUT     00H
28A4 C3 88 28       JMP     SUSI
                    END
```

Lösungen zum Abschnitt 4.4.

1.Aufgabe:

KIM

```
                    *=$0300
0300 A9 FF  START   LDA     #$FF
0302 8D 01 17       STA     $1701
0305 A9 00          LDA     #$00
0307 8D 00 17       STA     $1700
030A A2 A1  SUSI    LDX     #$A1
030C CA     RUDI    DEX
030D EA             NOP
030E D0 FC          BNE     RUDI
0310 EE 00 17       INC     $1700
0313 4C 0A 03       JMP     SUSI
                    .END
```

6800

```
                    ORG     $0180
0180 86 FF  START   LDA A   #$FF
0182 B7 80 04       STA A   $8004
0185 86 04          LDA A   #$04
0187 B7 80 05       STA A   $8005
018A 7F 80 04       CLR     $8004
018D 86 71  DORA    LDA A   #$71
018F 4A     HANS    DEC A
0190 26 FD          BNE     HANS
0192 7C 80 04       INC     AUS
0195 7E 01 8D       JMP     DORA
                    END
```

8080/8085

```
                    ORG     2880H
2880 3E FF  START   MVI     A,0FFH
2882 D3 02          OUT     02H
2884 3E 00          MVI     A,00H
2886 D3 00          OUT     00H
2888 06 E3  SUSI    MVI     B,0E3H
288A 05     MAX     DCR     B
288B C2 8A 28       JNZ     MAX
288E 3C             INR     A
288F D3 00          OUT     00H
2891 C3 88 28       JMP     SUSI
                    END
```

2.Aufgabe:

KIM

```
                    *=$0300
0300 A9 FF  START   LDA   #$FF
0302 8D 01 17       STA   $1701
0305 A9 00          LDA   #$00
0307 8D 03 17       STA   $1703
030A A2 01  MAUS    LDY   #$01
030C 8E 00 17 HUND  STX   $1700
030F E8             INX
0310 AD 02 17 PAUL  LDA   $1702
0313 30 FB          BMI   PAUL
0315 E0 07          CPX   #$07
0317 F0 F1          BEQ   MAUS
0319 4C 0C 03       JMP   HUND
                    END
```

6800

```
                    ORG   $0180
0180 86 FF  START   LDA A #$FF
0182 B7 80 04       STA A $8004
0185 86 04          LDA A #$04
0187 B7 80 05       STA A $8005
018A B7 80 07       STA A $8007
018D C6 01  MONI    LDA B #$01
018F F7 80 04 EVA   STA B $8004
0192 5C             INC B
0193 B6 80 06 LILO  LDA A $8006
0196 2B FB          BMI   LILO
0198 C1 07          CMP B #$07
019A 24 F1          BCC   MONI
019C 20 F1          BRA   EVA
                    END
```

8080/8085

```
                    ORG   2880H
2880 3E FF  START   MVI   A,0FFH
2882 D3 02          OUT   02H
2884 3E 00          MVI   A,00H
2886 D3 03          OUT   03H
2888 06 01  BERT    MVI   B,01H
288A 78             MOV   A,B
288B D3 00  HUND    OUT   00H
288D 04             INR   B
288E DB 01  MAX     IN    01H
2890 C6 00          ADI   00H
2892 FA 8E 28       JM    MAX
2895 78             MOV   A,B
2896 FE 07          CPI   07H
2898 CA 88 28       JZ    BERT
289B C3 8B 28       JMP   HUND
                    END
```

3.Aufgabe:

KIM

```
                    *=$0300
0300 A9 FF  START   LDA   #$FF
0302 8D 01 17       STA   $1701
0305 A9 00          LDA   #$00
0307 8D 03 17       STA   $1703
030A A9 00  PAULA   LDA   #$00
030C 8D 00 17       STA   $1700
030F A0 D9          LDY   #$D9
0311 A2 FF  PAPA    LDX   #$FF
0313 CA     MAMA    DEX
0314 EA             NOP
0315 EA             NOP
0316 D0 FB          BNE   MAMA
0318 88             DEY
0319 D0 F6          BNE   PAPA
031B AD 02 17       LDA   $1702
031E 8D 00 17       STA   $1700
0321 A9 D9          LDY   #$D9
0323 A2 FF  LOTTE   LDX   #$FF
0325 CA     KARL    DEX
```

6800

```
                    ORG   $0180
0180 86 FF  START   LDA A #$FF
0182 B7 80 04       STA A $8004
0185 86 04          LDA A #$04
0187 B7 80 05       STA A $8005
018A B7 80 07       STA A $8007
018D 7F 80 04 HUGO  CLR   $8004
0190 CE 96 00       LDX   #$9600
0193 09     MOPS    DEX
0194 26 FD          BNE   MOPS
0196 B6 80 06       LDA A $8006
0199 B7 80 04       STA A $8004
019C CE 96 00       LDX   #$9600
019F 09     PUPS    DEX
01A0 26 FD          BNE   PUPS
01A2 7E 01 8D       JMP   HUGO
                    END
```

8080/8085

```
                    ORG   2880H
2880 3E FF  START   MVI   A,0FFH
2882 D3 02          OUT   02H
2884 3E 00          MVI   A,00H
2886 D3 03          OUT   03H
2888 3E 00  HAXE    MVI   A,00H
288A D3 00          OUT   00H
288C 11 9B F6       LXI   D,0F69BH
288F 1B     MAUS    DCX   D
2890 7A             MOV   A,D
2891 B3             ORA   E
2892 C2 8F 28       JNZ   MAUS
2895 DB 01          IN    01H
2897 D3 00          OUT   00H
2899 11 9B F6       LXI   D,0F69BH
289C 1B     PAPA    DCX   D
289D 7A             MOV   A,D
289E B3             ORA   E
289F C2 9C 28       JNZ   PAPA
28A2 C3 88 28       JMP   HAXE
                    END
```

```
0326  EA        NOP
0327  EA        NOP
0328  DO        BNE     KARL
      FB
032A  88        DEY
032B  DO        BNE     LOTTE
      F6
032D  4C        JMP     PAULA
      0A
      03
                .END
```

Lösungen zum Abschnitt 4.5. und 4.6.

1.Aufgabe:

KIM

```
                *=$0000
0000  00  ADR   .WORD   $0200
      02
                *=$0300
0300  A9  START LDA     #$00
      00
0302  A8        TAY
0303  91  MONI  STA     (ADR),Y
      00
0305  C8        INY
0306  18        CLC
0307  69        ADC     #$01
      01
0309  DO        BNE     MONI
      F8
030B  EA        NOP
030C  A9        LDA     #$00
      00
030E  AA        TAX
030F  A8        TAY
0310  D1  MIMI  CMP     (ADR),Y
      00
0312  FO        BEQ     RENI
      01
0314  E8        INX
0315  C8  RENI  INY
0316  18        CLC
0317  69        ADC     #$01
      01
0319  DO        BNE     MIMI
      F5
031B  AO        LDY     #$00
      00
031D  8A        TXA
031E  91        STA     (ADR),Y
      00
0320  4C  MAUSI JMP     MAUSI
      20
      03
                .END
```

6800

```
                ORG     $0180
0180  00  ADR   FDB     $0000
      00
0182  FE  START LDX     ADR
      01
      80
0185  4F        CLR A
0186  A7  HUGO  STA A   00,X
      00
0188  08        INX
0189  4C        INC A
018A  26        BNE     HUGO
      FA
018C  01        NOP
018D  FE        LDX     ADR
      01
      80
0190  4F        CLR A
0191  5F        CLR B
0192  A1  OTTO  CMP A   00,X
      00
0194  27        BEQ     GUT
      01
0196  5C        INC B
0197  08  GUT   INX
0198  4C        INC A
0199  26        BNE     OTTO
      F7
019B  FE        LDX     ADR
      01
      80
019E  E7        STA B   00,X
      00
01A0  20  ENDE  BRA     ENDE
      FE
                END
```

8080/8085

```
                ORG     2880H
2880  00  ADR   DW      2000H
      20
2882  2A  START LHLD    ADR
      80
      28
2885  3E        MVI     A,00H
      00
2887  77  PAUL  MOV     M,A
2888  23        INX     H
2889  3C        INR     A
288A  C2        JNZ     PAUL
      87
      28
288D  00        NOP
288E  2A        LHLD    ADR
      80
      28
2891  3E        MVI     A,00H
      00
2893  47        MOV     B,A
2894  BE  OTTI  CMP     M
2895  CA        JZ      ABBA
      99
      28
2898  04        INR     B
2899  23  ABBA  INX     H
289A  3C        INR     A
289B  C2        JNZ     OTTI
      94
      28
289E  2A        LHLD    ADR
      80
      28
28A1  70        MOV     M,B
28A2  C3  HUGO  JMP     HUGO
      A2
      28
                END
```

2.Aufgabe:

KIM

```
                    *=$0000
0000        ADR     *=*+1
0001   02   ADRH    .BYTE   $02

                    *=$0300
0300   A9   START   LDA     #$FF
       FF
0302   8D           STA     $1701
       01
       17
0305   A9           LDA     #$00
       00
0307   8D           STA     $1703
       03
       17
030A   A8           TAY
030B   AD   RENI    LDA     $1703
       03
       17
030E   85           STAZ    ADR
       00
0310   B1           LDA     (ADR),Y
       00
0312   8D           STA     $1700
       00
       17
0315   4C           JMP     RENI
       0B
       03

                    .END
```

6800

```
                    ORG     $0180
0180   00   ADR1    FCB     00
0181        ADR2    RMB     1
0182   86   START   LDA A   #$FF
       FF
0184   B7           STA A   $8004
       80
       04
0187   86           LDA A   #$04
       04
0189   B7           STA A   $8005
       80
       05
018C   B7           STA A   $8007
       80
       07
018F   B6   PIPI    LDA A   $8006
       80
       06
0192   B7           STA A   ADR2
       01
       81
0195   FE           LDX     ADR1
       01
       80
0198   A6           LDA A   00,X
       00
019A   B7           STA A   $8004
       80
       04
019D   7E           JMP     PIPI
       01
       8F

                    END
```

8080/8085

```
                    ORG     2880H
2880   00   ADR     DW      2000H
       20
2882   3E   START   MVI     A,0FFH
       FF
2884   D3           OUT     02H
       02
2886   3E           MVI     A,00H
       00
2888   D3           OUT     03H
       03
288A   2A           LHLD    ADR
       80
       28
288D   DB   MAMA    IN      01H
       01
288F   6F           MOV     L,A
2890   7E           MOV     A,M
2891   D3           OUT     00H
       00
2893   C3           JMP     MAMA
       8D
       28

                    END
```

3.Aufgabe:

KIM

```
                    *=$0000
0000   00   ANF     .WORD   $0200
       02
0002   FF   END     .WORD   $02FF
       02

                    *=$0300
0300   A9   START   LDA     #$00
       00
0302   8A           TAY
0303   91   KERI    STA     (ANF),Y
       00
0305   C8           INY
0306   A6           LDXZ    ANF
       00
0308   E4           CPXZ    END
       02
030A   D0           BNE     KERI
       F7
030C   A6           LDXZ    ANF+1
       01
030E   E4           CPXZ    END+1
       03
0310   D0           BNE     KERI
       F1
0312   4C   REGI    JMP     REGI
       12
       03

                    .END
```

6800

```
                    ORG     $0180
0180   00   ANF     FDB     $0000
       00
0182   01   END     FDB     $0180
       80
0184   FE   START   LDX     ANF
       01
       80
0187   6F   BOBO    CLR     00,X
       00
0189   08           INX
018A   BC           CPX     END
       01
       82
018D   26           BNE     BOBO
       F8
018F   20   OTTO    BRA     OTTO
       FE

                    END
```

8080/8085

```
                    ORG     2880H
2880   00   ANF     DW      2000H
       20
2882   C7   END     DW      20C7
       20
2884   06   START   MVI     B,00H
       00
2886   2A           LHLD    END
       82
       28
2889   EB           XCHG
288A   2A           LHLD    ANF
       80
       28
288D   70   HELI    MOV     M,B
288E   23           INX     H
288F   7C           MOV     A,H
2890   BA           CMP     D
2891   C2           JNZ     HELI
       8D
       28
2894   7D           MOV     A,L
2895   BB           CMP     E
2896   C2           JNZ     HELI
       8D
       28
2899   C3   MAXI    JMP     MAXI
       99
       28

                    END
```

Lösungen zum Abschnitt 5.

1.Aufgabe:

KIM

```
                    *=$0300
0300  A9   START  LDA    #$FF
      FF
0302  8D          STA    $1701
      01
      17
0305  A9          LDA    #$00
      00
0307  8D          STA    $1703
      03
      17
030A  8D          STA    $1700
      00
      17
030D  A9          LDA    #$0F
      0F
030F  2C   PAPI   BIT    $1702
      02
      17
0312  F0          BEQ    PAPI
      FB
0314  CE          DEC    $1700
      00
      17
0317  A0          LDY    #$D9
      D9
0319  A2   HUGO   LDX    #$FF
      FF
031B  EA   MAMA   NOP
031C  EA          NOP
031D  CA          DEX
031E  D0          BNE    MAMA
      FB
0320  88          DEY
0321  D0          BNE    HUGO
      F6
0323  EE          INC    $1700
      00
      17
0326  A0          LDY    #$D9
      D9
0328  A2   MAX    LDX    #$FF
      FF
032A  EA   FILO   NOP
032B  EA          NOP
032C  CA          DEX
032D  D0          BNE    FILO
      FB
032F  88          DEY
0330  D0          BNE    MAX
      F6
0332  4C          JMP    PAPI
      0F
      03
                  .END
```

6800

```
                    ORG    $0180
0180  86   START  LDA A  #$FF
      FF
0182  B7          STA A  $8004
      80
      04
0185  86          LDA A  #$04
      04
0187  B7          STA A  $8005
      80
      05
018A  B7          STA A  $8007
      80
      07
018D  86          LDA A  #$0F
      0F
018F  7F          CLR    $8004
      80
      04
0192  B5   PAPA   BIT A  $8006
      80
      06
0195  27          BEQ    PAPA
      FB
0197  73          COM    $8004
      80
      04
019A  CE          LDX    #$9600
      96
      00
019D  09   PEPE   DEX
019E  26          BNE    PEPE
      FD
01A0  73          COM    $8004
      80
      04
01A3  CE          LDX    #$9600
      96
      00
01A6  09   PIPI   DEX
01A7  26          BNE    PIPI
      FD
01A9  7E          JMP    PAPA
      01
      92
                  END
```

8080/8085

```
                    ORG    2880H
2880  3E   START  MVI    A,0FFH
      FF
2882  D3          OUT    02H
      02
2884  3E          MVI    A,00H
      00
2886  D3          OUT    03H
      03
2888  DB   OLGA   IN     01H
      01
288A  E6          ANI    0FH
      0F
288C  C2          JZ     BLIN
      95
      28
288F  B7          ORA    A
2890  D3          OUT    00H
      00
2892  C3          JMP    OLGA
      88
      28
2895  3E   BLIN   MVI    A,0FFH
      FF
2897  D3          OUT    00H
      00
2899  11          LXI    D,0F69BH
      9B
      F6
289C  1B   MAUS   DCX    D
289D  7A          MOV    A,D
289E  B3          ORA    E
289F  C2          JNZ    MAUS
      9C
      28
28A2  3E          MVI    A,00H
      00
28A4  D3          OUT    00H
      00
28A6  11          LXI    D,0F69BH
      9B
      F6
28A9  1B   OTTI   DCX    D
28AA  7A          MOV    A,D
28AB  B3          ORA    E
28AC  C2          JNZ    OTTI
      A9
      28
28AF  C3          JMP    OLGA
      88
      28
                  END
```

2.Aufgabe:

KIM

```
                    *=$0300
0300  A9   START  LDA    #$FF
      FF
0302  8D          STA    $1701
      01
      17
0305  A9          LDA    #$00
      00
0307  8D          STA    $1703
      03
      17
```

6800

```
                    ORG    $0180
0180  86   START  LDA A  #$FF
      FF
0182  B7          STA A  $8004
      80
      04
0185  86          LDA A  #$04
      04
0187  B7          STA A  $8005
      80
      05
```

8080/8085

```
                    ORG    2880H
2880  3E   START  MVI    A,0FFH
      FF
2882  D3          OUT    02H
      02
2884  3E          MVI    A,00H
      00
2886  D3          OUT    03H
      03
2888  DB   POPO   IN     01H
      01
```

```
030A AD 02 17  BUBI  LDA   $1702
030D F0 20     BEQ   AUS
030F C9 90     CMP   #$90
0311 D0 05     BNE   HANS
0313 A9 55     LDA   #$55
0315 4C 2F 03  JMP   AUS
0318 C9 A4     HANS  CMP   #$A4
031A D0 05     BNE   GRETE
031C A9 AA     LDA   #$AA
031E 4C 2F 03  JMP   AUS
0321 C9 01     GRETE CMP   #$01
0323 F0 08     BEQ   FOX
0325 C9 02     CMP   #$02
0327 F0 04     BEQ   FOX
0329 C9 03     CMP   #$03
032B D0 08     BNE   FEHL
032D A9 FF     FOX   LDA   #$FF
032F 8D 00 17  AUS   STA   $1700
0332 4C 0A 03  JMP   BUBI
0335 A9 FF     FEHL  LDA   #$FF
0337 8D 00 17  STA   $1700
033A A0 D9     WALDI LDY   #$D9
033C A2 FF     HUGO  LDX   #$FF
033E EA        MAMA  NOP
033F EA              NOP
0340 CA              DEX
0341 D0 FB           BNE   MAMA
0343 88              DEY
0344 D0 F6           BNE   HUGO
0346 EE 00 17        INC   $1700
0349 F0 EF           BEQ   WALDI
034B 4C 0A 03        JMP   BUBI
                     .END
```

```
018A B7 80 07  STA A $8007
018D B6 80 06  LDA A $8006
0190 26 06     BNE   M2
0192 7F 80 04  CLR   $8004
0195 7E 01 8D  JMP   M1
0198 81 90     M2    CMP A #$90
019A 26 08     BNE   M3
019C 86 55     LDA A #$55
019E B7 80 04  STA A $8004
01A1 7E 01 8D  JMP   M1
01A4 81 A4     M3    CMP A #$A4
01A6 26 08     BNE   M4
01A8 86 AA     LDA A #$AA
01AA B7 80 04  STA A $8004
01AD 7E 01 8D  JMP   M1
01B0 81 01     M4    CMP A #$01
01B2 27 08     BEQ   M44
01B4 81 02     CMP A #$02
01B6 27 04     BEQ   M44
01B8 81 03     CMP A #$03
01BA 26 08     BNE   M5
01BC 86 FF     M44   LDA A #$FF
01BE B7 80 04  STA A $8004
01C1 7E 01 8D  JMP   M1
01C4 7F 80 04  M5    CLR   $8004
01C7 CE 96 00  LDX   #$9600
01CA 09        M6    DEX
01CB 26 FD     BNE   M6
01CD 73 80 04  COM   $8004
01D0 CE 96 00  LDX   #$9600
01D3 09        M7    DEX
01D4 26 FD     BNE   M7
01D6 7E 01 8D  JMP   M1
               END
```

```
288A C6 00     ADI   00H
288C CA B4 28  JZ    ABBA
288F FE 90     CPI   90H
2891 C2 99 28  JNZ   DORA
2894 3E 55     MVI   A,55H
2896 C3 B4 28  JMP   ABBA
2899 FE A4     DORA  CPI   0A4H
289B C2 A3 28  JNZ   ERNA
289E 3E AA     MVI   A,0AAH
28A0 C3 B4 28  JMP   ABBA
28A3 FE 01     ERNA  CPI   01H
28A5 CA B2 28  JZ    PINK
28A8 FE 02     CPI   02H
28AA CA B2 28  JZ    PINK
28AD FE 03     CPI   03H
28AF C2 B9 28  JNZ   FLOP
28B2 3E FF     PINK  MVI   A,0FFH
28B4 D3 00     ABBA  OUT   00H
28B6 C3 88 28  JMP   POPO
28B9 3E FF     FLOP  MVI   A,0FFH
28BB D3 00     MAJA  OUT   00H
28BD 67        MOV   H,A
28BE 11 9B F6  LXI   D,0F69BH
28C1 1B        FLIP  DCX   D
28C2 7A        MOV   A,D
28C3 B3        ORA   E
28C4 C2 C1 28  JNZ   FLIP
28C7 7C        MOV   A,H
28C8 3C        INR   A
28C9 CA BB 28  JZ    MAJA
28CC C3 88 28  JMP   POPO
               END
```

3.Aufgabe:

KIM

```
                    *=$0000
0000  00   TAB     .BYTE  $00
0001  FF           .BYTE  $FF
0002  01           .BYTE  $01
0003  02           .BYTE  $02
0004  03           .BYTE  $03
0005  04           .BYTE  $04
0006  05           .BYTE  $05
0007  06           .BYTE  $06
0008  07           .BYTE  $07

                    *=$0300
0300  A9   START   LDA    #$FF
      FF
0302  8D           STA    $1701
      01
      17
0305  A9           LDA    #$00
      00
0307  8D           STA    $1703
      03
      17
030A  AD   DIDI    LDA    $1702
      02
      17
030D  D0           BNE    MOLLI
      05
030F  A5           LDAZ   TAB
      00
0311  4C           JMP    INA
      1C
      03
0314  A2   MOLLI   LDX    #$09
      09
0316  CA   TOBI    DEX
0317  0A           ASLA
0318  90           BCC    TOBI
      FC
031A  B5           LDAZ   TAB,X
      00
031C  8D   INA     STA    $1700
      00
      17
031F  4C           JMP    DIDI
      0A
      03
                   .END
```

6800

```
                    ORG    $0000
0000  00   TAB     FCB    $00
0001  07           FCB    $07
0002  06           FCB    $06
0003  05           FCB    $05
0004  04           FCB    $04
0005  03           FCB    $03
0006  02           FCB    $02
0007  01           FCB    $01
0008  FF           FCB    $FF

                    ORG    $0180
0180  73   START   COM    $8004
      80
      04
0183  86           LDA A  #$04
      04
0185  B7           STA A  $8005
      80
      05
0188  B7           STA A  $8007
      80
      07
018B  CE   MIMI    LDX    #TAB
      00
      00
018E  B6           LDA A  $8006
      80
      06
0191  26           BNE    OLLI
      04
0193  A6           LDA A  00,X
      00
0195  20           BRA    MAXI
      06
0197  08   OLLI    INX
0198  48           ASL A
0199  24           BCC    OLLI
      FC
019B  A6           LDA A  00,X
      00
019D  B7   MAXI    STA A  $8004
      80
      04
01A0  20           BRA    MIMI
      E9
                   END
```

8080/8085

```
                    ORG    2000H
2000  00   TAB     DB     00H
2001  07           DB     07H
2002  06           DB     06H
2003  05           DB     05H
2004  04           DB     04H
2005  03           DB     03H
2006  02           DB     02H
2007  01           DB     01H
2008  FF           DB     0FFH

                    ORG    2880H
2880  3E   START   MVI    A,0FFH
      FF
2882  D3           OUT    02H
      02
2884  3E           MVI    A,00H
      00
2886  D3           OUT    03H
      03
2888  21   FIFI    LXI    H,TAB
      00
      20
288B  DB           IN     01H
      01
288D  C6           ADI    00H
      00
288F  C2           JNZ    OMA
      96
      28
2892  77           MOV    A,M
2893  C3           JMP    BEN
      9C
      28
2896  23   OMA     INX    H
2897  17           RAL
2898  D2           JNC    OMA
      96
      28
289B  7E           MOV    A,M
289C  D3   BEN     OUT    00H
      00
289E  C3           JMP    FIFI
      88
      28
                   END
```

Lösungen zum Abschnitt 6.

1.Aufgabe:

KIM

```
                    *=$0300
0300 A9 00  START   LDA   #$00
0302 8D 03 17       STA   $1703
0305 A9 FF          LDA   #$FF
0307 8D 01 17       STA   $1701
030A AD 02 17 RENI  LDA   $1702
030D 49 03          EOR   #$03
030F 29 CF          AND   #$CF
0311 09 C0          ORA   #$C0
0313 8D 00 17       STA   $1700
0316 4C 0A 03       JMP   RENI
                    .END
```

6800

```
                    ORG   $0180
0180 73 80  START   COM   $8004
0183 86 04          LDA A #$04
0185 B7 80 05       STA A $8005
0188 B7 80 07       STA A $8007
018B B6 80 06 LILA  LDA A $8006
018E 88 03          EOR A #$03
0190 84 CF          AND A #$CF
0192 8A C0          ORA A #$C0
0194 B7 80 04       STA A $8004
0197 7E 01 8B       JMP   LILA
                    END
```

8080/8085

```
                    ORG   2880H
2880 3E FF  START   MVI   A,0FFH
2882 D3 02          OUT   02H
2884 3E 00          MVI   A,00H
2886 D3 03          OUT   03H
2888 DB 01  LILO    IN    01H
288A EE 03          XRI   03H
288C E6 CF          ANI   0CFH
288E F6 C0          ORI   0C0H
2890 D3 00          OUT   00H
2892 C3 88 28       JMP   LILO
                    END
```

2.Aufgabe:

KIM

```
                    *=$0300
0300 A9 00  START   LDA   #$00
0302 8D 03 17       STA   $1703
0305 A9 FF          LDA   #$FF
0307 8D 01 17       STA   $1701
030A D8             CLD
030B AD 02 17 KATI  LDA   $1702
030E A8             TAY
030F 4A             LSRA
0310 4A             LSRA
0311 4A             LSRA
0312 4A             LSRA
0313 F0 0E          BEQ   FINA
0315 AA             TAX
0316 98             TYA
0317 29 0F          AND   #$0F
0319 85 00          STAZ  $00
031B 18             CLC
031C A9 00          LDA   #$00
031E 65 00  MARY    ADCZ  $00
0320 CA             DEX
0321 D0 FB          BNE   MARY
0323 8D 00 17 FINA  STA   $1700
0326 4C 0B 03       JMP   KATI
                    .END
```

6800

```
                    ORG   $0180
0180 73 80  START   COM   $8004
0183 86 04          LDA A #$04
0185 B7 80 05       STA A $8005
0188 B7 80 07       STA A $8007
018B B6 80 06 LEO   LDA A $8006
018E 16             TAB
018F 44             LSR A
0190 44             LSR A
0191 44             LSR A
0192 44             LSR A
0193 27 0C          BEQ   AUS
0195 97 00          STAZ  $00
0197 4F             CLR A
0198 C4 0F          AND B #$0F
019A 27 05  DORA    BEQ   AUS
019C 9B 00          ADDZ A $00
019E 5A             DEC B
019F 20 F9          BRA   DORA
01A1 B7 80 04 AUS   STA A $8004
01A4 7E 01 8B       JMP   LEO
                    END
```

8080/8085

```
                    ORG   2880H
2880 3E FF  START   MVI   A,0FFH
2882 D3 02          OUT   02H
2884 3E 00          MVI   A,00H
2886 D3 03          OUT   03H
2888 DB 01  LEO     IN    01H
288A 47             MOV   B,A
288B 0F             RRC
288C 0F             RRC
288D 0F             RRC
288E 0F             RRC
288F E6 0F          ANI   0FH
2891 4F             MOV   C,A
2892 78             MOV   A,B
2893 E6 0F          ANI   0FH
2895 47             MOV   B,A
2896 AF             XRA   A
2897 B9             CMP   C
2898 CA A0 28 DORA  JZ    AUS
289B 80             ADD   B
289C 0D             DCR   C
289D C3 98 28       JMP   DORA
28A0 D3 00  AUS     OUT   00H
28A2 C3 88 28       JMP   LEO
                    END
```

3.Aufgabe:

KIM

```
                    *=$0300
0300 A9     START   LDA     #$00
     00
0302 8D             STA     $1703
     03
     17
0305 A9             LDA     #$FF
     FF
0307 8D             STA     $1701
     01
     17
030A D8             CLD
030B AD     INES    LDA     $1702
     02
     17
030E A2             LDX     #$00
     00
0310 C9     PAUL    CMP     #$0A
     0A
0312 90             BCC     MAUS
     06
0314 E9             SBC     #$0A
     0A
0316 E8             INX
0317 4C             JMP     PAUL
     10
     03
031A 86     MAUS    STXZ    $00
     00
031C 06             ASLZ    $00
     00
031E 06             ASLZ    $00
     00
0320 06             ASLZ    $00
     00
0322 06             ASLZ    $00
     00
0324 05             ORAZ    $00
     00
0326 8D             STA     $1700
     00
     17
0329 4C             JMP     INES
     0B
     03
                    .END
```

6800

```
                    ORG     $0180
0180 73     START   COM     $8004
     80
     04
0183 86             LDA A   #$04
     04
0185 B7             STA A   $8005
     80
     05
0188 B7             STA A   $8007
     80
     07
018B B6     HUND    LDA A   $8006
     80
     06
018E 5F             CLR B
018F 81     ANNE    CMP A   #$0A
     0A
0191 25             BCS     MAUS
     05
0193 80             SUB A   #$0A
     0A
0195 5C             INC B
0196 20             BRA     ANNE
     F7
0198 58     MAUS    ASL B
0199 58             ASL B
019A 58             ASL B
019B 58             ASL B
019C 1B             ABA
019D B7             STA A   $8004
     80
     04
01A0 7E             JMP     HUND
     01
     8B
                    END
```

8080/8085

```
                    ORG     2880H
2880 3E     START   MVI     A,0FFH
     FF
2882 D3             OUT     02H
     02
2884 3E             MVI     A,00H
     00
2886 D3             OUT     03H
     03
2888 DB     FEE     IN      01H
     01
288A 06             MVI     B,00H
     00
288C FE     RITA    CPI     0AH
     0A
288E DA             JC      MOLLI
     97
     28
2891 D6             SUI     0AH
     0A
2893 04             INR     B
2894 C3             JMP     RITA
     8C
     28
2897 4F     MOLLI   MOV     C,A
2898 78             MOV     A,B
2899 07             RLC
289A 07             RLC
289B 07             RLC
289C 07             RLC
289D B1             ORA     C
289E D3             OUT     00H
     00
28A0 C3             JMP     FEE
     88
     28
                    END
```

Lösungen zum Abschnitt 7.

1.Aufgabe:

KIM

```
                    *=$0300
0300 A9     START   LDA     #$00
     00
0302 8D             STA     $1703
     03
     17
0305 A9             LDA     #$FF
     FF
0307 8D             STA     $1701
     01
     17
030A A9             LDA     #$01
     01
030C 8D             STA     $1700
     00
     17
030F 18             CLC
0210 AD     SUSI    LDA     $1702
     02
     17
```

6800

```
                    ORG     $0180
0180 73     START   COM     $8004
     80
     04
0183 86             LDA A   #$04
     04
0185 B7             STA A   $8005
     80
     05
0188 B7             STA A   $8007
     80
     07
018B 86             LDA A #$01
     01
018D B7             STA A   $8004
     80
     04
0190 0C             CLC
0191 B6     MOPS    LDA A   $8006
     80
```

8080/8085

```
                    ORG     2880H
2880 3E     START   MVI     A,0FFH
     FF
2882 D3             OUT     02H
     02
2884 3C             INR     A
2885 D3             OUT     03H
     03
2887 3C             INR     A
2888 D3             OUT     00H
     00
288A 47             MOV     B,A
288B DB     STAN    IN      01H
     01
288D CD             CALL    WARTE
     A0
     28
2890 78             MOV     A,B
2891 07             RLC
2892 D3             OUT     00H
```

```
0313  20          JSR    WARTE
      20
      03
0316  2E   VIVI   ROL    $1700
      00
      17
0319  B0          BCS    VIVI
      FB
031B  4C          JMP    SUSI
      10
      03

                  *=$0320
0320  AA   WARTE  TAX
0321  F0   IRMA   BEQ    RAUS
      0B
0323  A0          LDY    #$DD
      DD
0325  88   ANNE   DEY
0326  EA          NOP
0327  EA          NOP
0328  D0          BNE    ANNE
      FB
032A  CA          DEX
032B  4C          JMP    IRMA
      21
      03
032E  60   RAUS   RTS
                  END
```

```
      06
0194  BD          JSR    WARTE
      01
      A0
0197  79   OLGA   ROL    $8004
      80
      04
019A  25          BCS    OLGA
      FB
019C  7E          JMP    MOPS
      01
      91

                  ORG    $01A0
01A0  37   WARTE  PSH B
01A1  4D          TST A
01A2  27   OMA    BEQ    DIDI
      08
01A4  C6          LDA B  #$CC
      CC
01A6  5A   PAPA   DEC B
01A7  26          BNE    PAPA
      FD
01A9  4A          DEC A
01AA  20          BRA    OMA
      F6
01AC  33   DIDI   PUL B
01AD  39          RTS
                  END
```

```
      00
2894  47          MOV    B,A
2895  C3          JMP    STAN
      8B
      28

                  ORG    28A0H
28A0  D5   WARTE  PUSH   D
28A1  C6          ADI    00H
      00
28A3  CA   OLLI   JZ     MAMA
      B3
      28
28A6  16          MVI    D,0E0H
      E0
28A8  15   RITA   DCR    D
28A9  00          NOP
28AA  00          NOP
28AB  00          NOP
28AC  C2          JNZ    RITA
      A8
      28
28AF  3D          DCR    A
28B0  C3          JMP    OLLI
      A3
      28
28B3  D1   MAMA   POP    D
28B4  C9          RET
                  END
```

2.Aufgabe:

KIM

```
                  *=$0300
0300  A9   START  LDA    #$00
      00
0302  8D          STA    $1703
      03
      17
0305  A9          LDA    #$FF
      FF
0307  8D          STA    $1701
      01
      17
030A  A9          LDA    #$20
      20
030C  8D          STA    $17FA
      FA
      17
030F  A9          LDA    #$03
      03
0311  8D          STA    $17FB
      FB
      17
0314  D8          CLD
0315  4C   OTTO   JMP    OTTO
      15
      03
```

6800

```
                  ORG    $0180
0180  73   START  COM    $8004
      80
      04
0183  86          LDA A  #$04
      04
0185  B7          STA A  $8005
      80
      05
0188  B7          STA A  $8007
      80
      07
018B  20   OTTO   BRA    OTTO
      FE
```

8080/8085

```
                  ORG    20D4H
20D4  C3   ADR    JMP    IDA
      90
      28

                  ORG    2880H
2880  AF   START  XRA    A
2881  D3          OUT    03H
      03
2883  3D          DCR    A
2884  D3          OUT    02H
      02
2886  3E          MVI    A,18H
      18
2888  30          SIM
2889  FB   BALL   EI
288A  76          HLT
288B  C3          JMP    BALL
      89
      28
```

```
                  *=$0320
0320  AD   HUGO   LDA    $1702
      02
      17
0323  20          JSR    DEDU
      30
      03
0326  8D          STA    $1700
      00
      17
0329  40          RTI
```

```
                  ORG    $01A0
01A0  B6   INES   LDA A  $8006
      80
      06
01A3  BD          JSR    DEDU
      01
      B0
01A6  B7          STA A  $8004
      80
      04
01A9  3B          RTI
```

```
                  ORG    2890H
2890  DB   IDA    IN     01H
      01
2892  CD          CALL   GIRL
      A0
      28
2895  D3          OUT    00H
      00
2897  C9          RET
```

```
                    *=$0330
0330  48    DEDU  PHA
0331  8A          TXA
0332  48          PHA
0333  BA          TSX
0334  BD          LDA     $0102,X
      02
      01
0337  29          AND     #$0F
      0F
0339  48          PHA
033A  BD          LDA     $0102,X
      02
      01
033D  4A          LSRA
033E  4A          LSRA
033F  4A          LSRA
0340  4A          LSRA
0341  9D          STA     $0102,X
      02
      01
0344  18          CLC
0345  2A          ROLA
0346  2A          ROLA
0347  7D          ADC     $0102,X
      02
      01
034A  2A          ROLA
034B  7D          ADC     $0100,X
      00
      01
034E  9D          STA     $0102,X
      02
      01
0351  68          PLA
0352  68          PLA
0353  AA          TAX
0354  68          PLA
0355  60          RTS
                  .END
```

```
                    ORG     $0180
0180  37    DEDU  PSH B
0181  36          PSH A
0182  44          LSR A
0183  44          LSR A
0184  44          LSR A
0185  44          LSR A
0186  16          TBA
0187  48          ASL A
0188  48          ASL A
0189  1B          ABA
018A  48          ASL A
018B  33          PUL B
018C  C4          AND B   #$0F
      0F
018E  1B          ABA
018F  33          PUL B
01C0  39          RTS
                  END
```

```
                    ORG     28A0H
28A0  C5    GIRL  PUSH    B
28A1  F5          PUSH    PSW
28A2  47          MOV     B,A
28A3  0F          RRC
28A4  0F          RRC
28A5  0F          RRC
28A6  0F          RRC
28A7  E6          ANI     0FH
      0F
28A9  4F          MOV     C,A
28AA  78          MOV     A,B
28AB  E6          ANI     0FH
      0F
28AD  47          MOV     B,A
28AE  79          MOV     A,C
28AF  07          RLC
28B0  07          RLC
28B1  81          ADD     C
28B2  07          RLC
28B3  80          ADD     B
28B4  47          MOV     B,A
28B5  F1          POP     PSW
28B6  78          MOV     A,B
28B7  C1          POP     B
28B8  C9          RET
                  END
```

3.Aufgabe:

KIM

```
                    *=$0300
0300  A9    START LDA     #$00
      00
0302  8D          STA     $1703
      03
      17
0305  A9          LDA     #$FF
      FF
0307  8D          STA     $1701
      01
      17
030A  A9          LDA     #$30
      30
030C  8D          STA     $17FA
      FA
      17
030F  A9          LDA     #$03
      03
0311  8D          STA     $17FB
      FB
      17
0314  A2          LDX     #$00
      00
0316  A0          LDY     #$01
      01
0318  AD    ANNI  LDA     $1702
      02
      17
031B  29          AND     #$0F
      0F
031D  8D          STA     $1700
      00
      17
0320  4C          JMP     ANNI
      18
      03
```

6800

```
                    ORG     $0180
0180  73    START COM      $8004
      80
      04
0183  86          LDA A   #$04
      04
0185  B7          STA A   $8005
      80
      05
0188  B7          STA A   $8007
      80
      07
018B  CE          LDX     #$0000
      00
      00
018E  0E          CLI
018F  B6    PAUL  LDA A   $8006
      80
      06
0192  B7          STA A   $8004
      80
      04
0195  7E          JMP     PAUL
      01
      8F
```

8080/8085

```
                    ORG     20D4H
20D4  C3    ADR   JMP     KAIN
      A0
      28
                    ORG     2880H
2880  AF    START XRA     A
2881  D3          OUT     03H
      03
2883  3D          DCR     A
2884  D3          OUT     02H
      02
2886  3E          MVI     A,1BH
      1B
2888  30          SIM
2889  21          LXI     H,2000H
      00
      20
288C  FB    NANA  EI
288D  76          HLT
288E  C3          JMP     NANA
      8C
      28
```

```
                *=$0330
0330  29  IRMA  AND    #$0F
      0F
0332  88        DEY
0333  D0        BNE    OLGA
      07
0335  0A        ASLA
0336  0A        ASLA
0337  0A        ASLA
0338  0A        ASLA
0339  95        STAZ   $00,X
      00
033B  40        RTI
033C  A0  OLGA  LDY    #$01
      01
033E  15        ORAZ   $00,X
      00
0340  95        STAZ   $00,X
      00
0342  E8        INX
0343  C9        CMP    #$AA
      AA
0345  F0        BEQ    RENI
      01
0347  40        RTI
0348  4C  RENI  JMP    $0000
      00
      00
                .END
```

```
                ORG    $01A0
01A0  A7  IRMA  STA A  00,X
      00
01A2  08        INX
01A3  81        CMP A  #$AA
      AA
01A5  27        BEQ    ANNI
      01
01A7  3B        RTI
01A8  7E  ANNI  JMP    $0000
      00
      00
                END
```

```
                ORG    28A0H
28A0  F3  KAIN  DI
28A1  DB        IN     01H
      01
28A3  FE        CPI    0AAH
      AA
28A5  CA        JZ     LOS
      B0
      28
28A8  77        MOV    M,A
28A9  23        INX    H
28AA  D3        OUT    00H
      00
28AC  3E        MVI    A,1BH
      1B
28AE  30        SIM
28AF  C9        RET
28B0  C3  LOS   JMP    2000H
      00
      20
                END
```

9. Ergänzende und weiterführende Literatur

9.1 Einführungen

1. Rechenberg, P.: Grundzüge digitaler Rechenautomaten. Oldenbourg München - Wien 1968

2. Schmid, D. Senger, D. Wojtkowiak, H.: Technische Informatik. Teil 1: Grundprinzipien des Entwurfs und der Organisation digitaler Rechenanlagen. Oldenbourg München - Wien 1973

3. Pelka, H.: Von der Schaltalgebra zum Mikroprozessor. Franzis-Verlag München 1977

4. Kobitzsch, W.: Mikroprozessoren - Aufbau und Wirkungsweise. Teil 1: Grundlagen Oldenbourg München - Wien 1977

5. Osborne, A.: Einführung in die Mikrocomputertechnik. te-wi Verlag München 1977

6. Texas Instr.: Einführung in die Mikroprozessortechnik. Firmenschrift Texas Instruments 1977

7. Dirks, C. Krinn, H.: Mikrocomputer Teil 1: Einführung Berliner Union Kohlhammer 1976

9.2 Anwendungen

8. Kobitzsch, W.: Mikroprozessoren - Aufbau und Wirkungsweise. Teil 2: Anwendungen und Beispiele Oldenbourg München - Wien 1978

9. Hilberg, W. Piloty, R.: Mikroprozessoren und ihre Anwendungen. Oldenbourg München - Wien 1977

10. ELEKTRONIK: Sonderheft Mikroprozessoren - Hardware Franzis-Verlag München 1977

11. ELEKTRONIK: Sonderheft Mikroprozessoren - Software Franzis-Verlag München 1977

12. Dirks, C. Krinn, H.: Mikrocomputer Teil 2: Anwendungen Berliner Union Kohlhammer 1978

13. FELTRON: Microcomputer - Informationen Lose-Blatt-Sammlung der Firma FELTRON-ELEKTRONIK Troisdorf

9.3 Literatur zum KIM-1

14. KIM-1 Handbuch
MCDS Microcomputer Datensysteme
Darmstadt 1977

15. KIM-1 Programmierhandbuch
MCDS Microcomputer Datensysteme
Darmstadt 1977

16. KIM-1 Hardwarehandbuch
MCDS Microcomputer Datensysteme
Darmstadt 1977

17. KIMath Subroutines Programming Manual
MCDS Microcomputer Datensysteme
Darmstadt

18. Softbook of KIM
MCDS Microcomputer Datensysteme
Darmstadt

19. KIM Assembler Manual
MCDS Microcomputer Datensysteme
Darmstadt

20. Cross Assembler Manual
MCDS Microcomputer Datensysteme
Darmstadt

9.4 Literatur zum Motorola 6800

21. MOTOROLA: M6800 Microcomputer System Design Data
Firmenschrift Motorola

22. MOTOROLA: M6800 Microprocessor Applications Manual
Firmenschrift Motorola

23. MOTOROLA: M6800 Microprozessor Programming Manual
Firmenschrift Motorola

9.5 Literatur zum INTEL 8080 bzw. 8085

24. Burghardt, K.: Der Mikroprozessor - Vom Bauteil zur Anwendung
Firmenschrift NEYE Enatechnik Quickborn

25. SIEMENS: Mikrocomputer SAB 8080 Programmierung
Firmenschrift SIEMENS München

26. SIEMENS: Datenbuch Mikroprozessor-Bausteine SAB 8080
Firmenschrift SIEMENS München

27. SIEMENS: Programmbibliothek SAB 8080
Band 1: Grundrechenarten
Band 2: Code-Umwandlungen
Firmenschrift SIEMENS München

28. Pelka, H.: Praxis mit Mikroprozessoren - Aufsatzreihe in
Funkschau H.15 (1977) bis H.10 (1978)
Franzis-Verlag München

10. Anhang

Anhang 1: Codes zur Darstellung von Dezimalziffern

Ziffer	8-4-2-1	Aiken 2-4-2-1	3-Excess	2-aus-5 7-4-2-1-0
0	0 0 0 0	0 0 0 0	0 0 1 1	1 1 0 0 0
1	0 0 0 1	0 0 0 1	0 1 0 0	0 0 0 1 1
2	0 0 1 0	0 0 1 0	0 1 0 1	0 0 1 0 1
3	0 0 1 1	0 0 1 1	0 1 1 0	0 0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 1 1	0 1 0 0 1
5	0 1 0 1	1 0 1 1	1 0 0 0	0 1 0 1 0
6	0 1 1 0	1 1 0 0	1 0 0 1	0 1 1 0 0
7	0 1 1 1	1 1 0 1	1 0 1 0	1 0 0 0 1
8	1 0 0 0	1 1 1 0	1 0 1 1	1 0 0 1 0
9	1 0 0 1	1 1 1 1	1 1 0 0	1 0 1 0 0

Anhang 2: ASCII-Zeichencode

HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	\	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5\	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	- ←	6F	o	7F	DEL

Anhang 3: Dezimal-Hexadezimal-Tabelle von -128 bis +127
negative Zahlen als 2er-Komplement

DEZ	HEX	KPL	DEZ	HEX	KPL	DEZ	HEX	KPL	DEZ	HEX	KPL
00	00	00	32	20	E0	64	40	C0	96	60	A0
01	01	FF	33	21	DF	65	41	BF	97	61	9F
02	02	FE	34	22	DE	66	42	BE	98	62	9E
03	03	FD	35	23	DD	67	43	BD	99	63	9D
04	04	FC	36	24	DC	68	44	BC	100	64	9C
05	05	FB	37	25	DB	69	45	BB	101	65	9B
06	06	FA	38	26	DA	70	46	BA	102	66	9A
07	07	F9	39	27	D9	71	47	B9	103	67	99
08	08	F8	40	28	D8	72	48	B8	104	68	98
09	09	F7	41	29	D7	73	49	B7	105	69	97
10	0A	F6	42	2A	D6	74	4A	B6	106	6A	96
11	0B	F5	43	2B	D5	75	4B	B5	107	6B	95
12	0C	F4	44	2C	D4	76	4C	B4	108	6C	94
13	0D	F3	45	2D	D3	77	4D	B3	109	6D	93
14	0E	F2	46	2E	D2	78	4E	B2	110	6E	92
15	0F	F1	47	2F	D1	79	4F	B1	111	6F	91
16	10	F0	48	30	D0	80	50	B0	112	70	90
17	11	EF	49	31	CF	81	51	AF	113	71	8F
18	12	EE	50	32	CE	82	52	AE	114	72	8E
19	13	ED	51	33	CD	83	53	AD	115	73	8D
20	14	EC	52	34	CC	84	54	AC	116	74	8C
21	15	EB	53	35	CB	85	55	AB	117	75	8B
22	16	EA	54	36	CA	86	56	AA	118	76	8A
23	17	E9	55	37	C9	87	57	A9	119	77	89
24	18	E8	56	38	C8	88	58	A8	120	78	88
25	19	E7	57	39	C7	89	59	A7	121	79	87
26	1A	E6	58	3A	C6	90	5A	A6	122	7A	86
27	1B	E5	59	3B	C5	91	5B	A5	123	7B	85
28	1C	E4	60	3C	C4	92	5C	A4	124	7C	84
29	1D	E3	61	3D	C3	93	5D	A3	125	7D	83
30	1E	E2	62	3E	C2	94	5E	A2	126	7E	82
31	1F	E1	63	3F	C1	95	5F	A1	127	7F	81
									128		80

Anhang 4: Dezimal-Hexadezimal-Tabelle von 0 bis 1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00_	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01_	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02_	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03_	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04_	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05_	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06_	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07_	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08_	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09_	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A_	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B_	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C_	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D_	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E_	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F_	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10_	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11_	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12_	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13_	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14_	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15_	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16_	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17_	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18_	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19_	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A_	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B_	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C_	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D_	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E_	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F_	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20_	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21_	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22_	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23_	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24_	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25_	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26_	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27_	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28_	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29_	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A_	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B_	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C_	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D_	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E_	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F_	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30_	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31_	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32_	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33_	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34_	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35_	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36_	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37_	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38_	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39_	0912	0913	0914	0915	0915	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A_	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B_	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C_	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D_	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E_	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F_	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

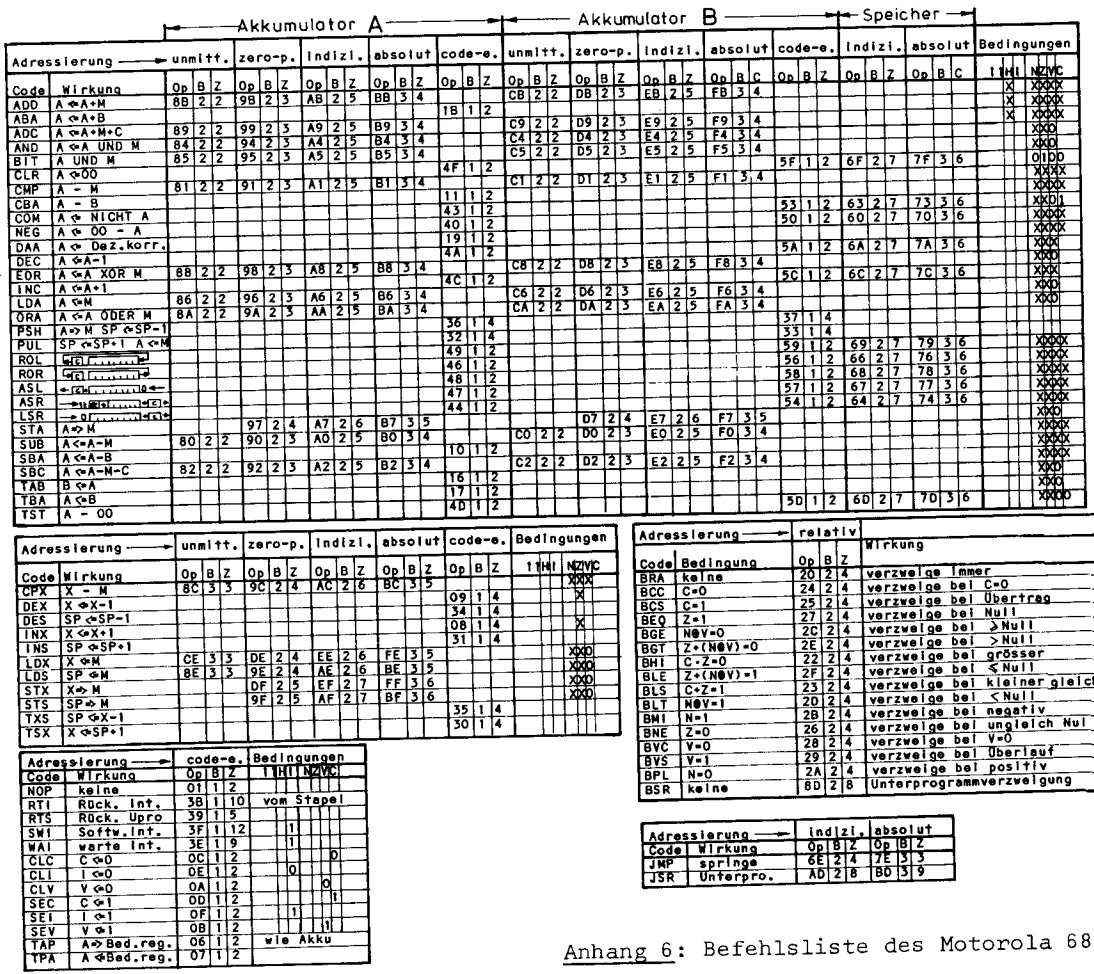
Anhang 5: Befehlsliste des KIM-1

Sym	Befehl	unmitt Op	C	B	absolut Op	C	B	zero-p. Op	C	B	Akkum. Op	C	B	code-e. Op	C	B	(In,X) Op	C	B	(In),Y Op	C	B	∠Pag,X Op	C	B	Abs,X Op	C	B	Abs,Y Op	C	B	Relat. Op	C	B	Indir. Op	C	B	ZPag,Y Op	C	B	N	Z	C	I	D	V
ADC	A+M+C→A	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										+	+	+			+
AND	A∧M→A	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										+	+				
ASL	←C 7 6 5 4 3 2 1 0←				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													+	+	+			
BCC	Spr C=0																															90	2	2											
BCS	Spr C=1																															B0	2	2											
BEQ	Spr Z=1																															F0	2	2											
BIT	A∧M				2C	4	3	24	3	2																														M7	+				M6
BMI	Spr N=1																															30	2	2											
BNE	Spr Z=0																															D0	2	2											
BPL	Spr N=0																															10	2	2											
BRK	Interrupt													00	7	1																											1		
BVC	Spr V=0																															50	2	2											
BVS	Spr V=1																															70	2	2											
CLC	0→C													18	2	1																											0		
CLD	0→D													D8	2	1																												C	
CLI	0→I													58	2	1																													
CLV	0→V													B8	2	1																													0
CMP	A - M	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										+	+	+			
CPX	X - M	E0	2	2	EC	4	3	E4	3	2																																+	+	+			
CPY	Y - M	C0	2	2	CC	4	3	C4	3	2																																+	+	+			
DEC	M-1→M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													+	+				
DEX	X-1→X													CA	2	1																									+	+				
DEY	Y-1→Y													88	2	1																									+	+				
EOR	A⊻M→A XOR	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										+	+				
INC	M+1→M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													+	+				
INX	X+1→X													E8	2	1																									+	+				
INY	Y+1→Y													C8	2	1																									+	+				
JMP	Sprung				4C	3	3																												6C	5	3									
JSR	Spr Upro				20	6	3																																							
LDA	M→A	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										+	+				
LDX	M→X	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	+	+				
LDY	M→Y	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													+	+				
LSR	→0 7 6 5 4 3 2 1 0→				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0	+	+			
NOP	No Oper.													EA	2	1																													
ORA	A∨M→A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										+	+				
PHA	A→Ms S-1→S													48	3	1																													
PHP	P→Ms S-1→S													08	3	1																													
PLA	S+1→S Ms→A													68	4	1																									+	+				(zurückgesp.)
PLP	S+1→S Ms→P													28	4	1																									+	+	+			(zurückgesp.)
ROL	C 7 6 5 4 3 2 1 0				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													+	+	+			
ROR	C 7 6 5 4 3 2 1 0				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													+	+	+			
RTI	Rückspr. In													40	6	1																													
RTS	Rückspr. Up													60	6	1																													
SBC	A-M-C→A	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										+	+				+
SEC	1→C													38	2	1																											1		
SED	1→D													F8	2	1																												1	
SEI	1→I													78	2	1																											1		
STA	A→M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3															
STX	X→M				8E	4	3	86	3	2													96	4	2																					
STY	Y→M				8C	4	3	84	3	2													94	4	2																					
TAX	A→X													AA	2	1																									+	+				
TAY	A→Y													A8	2	1																									+	+				
TSX	S→X													BA	2	1																									+	+				
TXA	X→A													8A	2	1																									+	+				
TXS	X→S													9A	2	1																													
TYA	Y→A													98	2	1																									+	+				

OP : Operationscode hexadezimal
C : Zahl der Zyklen bei der Ausführung
B : Zahl der Bytes des Befehls

\+ : Ergebnis setzt Bit
0 : Bit wird 0
1 : Bit wird 1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK	ORA(,X)				ORAZ	ASLZ		PHP	ORA #	ASLA			ORA	ASL	
1	BPL	ORA(),Y				ORAZ ,X	ASLZ ,X		CLC	ORA ,Y				ORA ,X	ASL ,X	
2	JSR	AND(,X)			BITZ	ANDZ	ROLZ		PLP	AND #	ROLA		BIT	AND	ROL	
3	BMI	AND(),Y				ANDZ ,X	ROLZ ,X		SEC	AND ,Y				AND ,X	ROL ,X	
4	RTI	EOR(,X)				EORZ	LSRZ		PHA	EOR #	LSRA		JMP	EOR	LSR	
5	BVC	EOR(),Y				EORZ ,X	LSRZ ,X		CLI	EOR ,Y				EOR ,X	LSR ,X	
6	RTS	ADC(,X)				ADCZ	RORZ		PLA	ADC #	RORA		JMP()	ADC	ROR	
7	BVS	ADC(),Y				ADCZ ,X	RORZ ,X		SEI	ADC ,Y				ADC ,X	ROR ,X	
8		STA(,X)			STYZ	STAZ	STXZ		DEY		TXA		STY	STA	STX	
9	BCC	STA(),Y			STYZ ,X	STAZ ,X	STXZ ,Y		TYA	STA ,Y	TXS			STA ,X		
A	LDY #	LDA(,X)	LDX #		LDYZ	LDAZ	LDXZ		TAY	LDA #	TAX		LDY	LDA	LDX	
B	BCS	LDA(),Y			LDYZ ,X	LDAZ ,X	LDXZ ,Y		CLV	LDA ,Y	TSX		LDY ,X	LDA ,X	LDX,Y	
C	CPY #	CMP(,X)			CPYZ	CMPZ	DECZ		INY	CMP #	DEX		CPY	CMP	DEC	
D	BNE	CMP(),Y				CMPZ ,X	DECZ ,X		CLD	CMP ,Y				CMP,X	DEC ,X	
E	CPX #	SBC(,X)			CPXZ	SBCZ	INCZ		INX	SBC #	NOP		CPX	SBC	INC	
F	BEQ	SBC(),Y				SBCZ ,X	INCZ ,X		SED	SBC ,Y				SBC ,X	INC ,X	



Adressierung →	Akkumulator A: unmitt.				zero-p.				indizl.				absolut				code-e.			Akkumulator B: unmitt.				zero-p.				indizl.				absolut				code-e.			Speicher: indizl.			absolut			Bedingungen					
Code	Wirkung	Op	B	Z	Op	B	Z		Op	B	Z		Op	B	Z		Op	B	Z	Op	B	Z		Op	B	Z		Op	B	Z		Op	B	C		Op	B	Z		Op	B	C	↕	H	I	N	Z	V	C
ADD	A ⇐ A+M	BB	2	2	9B	2	3		AB	2	5		BB	3	4					CB	2	2		DB	2	3		EB	2	5		FB	3	4												X	X	X	X	X
ABA	A ⇐ A+B																1B	1	2																													
ADC	A ⇐ A+M+C	89	2	2	99	2	3		A9	2	5		B9	3	4					C9	2	2		D9	2	3		E9	2	5		F9	3	4										X	X	X	X	X
AND	A ⇐ A UND M	84	2	2	94	2	3		A4	2	5		B4	3	4					C4	2	2		D4	2	3		E4	2	5		F4	3	4										X	X	0	X	0
BIT	A UND M	85	2	2	95	2	3		A5	2	5		B5	3	4					C5	2	2		D5	2	3		E5	2	5		F5	3	4										X	X	0	X	0
CLR	A ⇐ 00																4F	1	2																													
CMP	A - M	81	2	2	91	2	3		A1	2	5		B1	3	4					C1	2	2		D1	2	3		E1	2	5		F1	3	4						5F	1	2	6F	2	7	7F	3	6	X	X	X	X	X
CBA	A - B																11	1	2																													
COM	A ⇐ NICHT A																43	1	2																						53	1	2	63	2	7	73	3	6	X	X	0	1	1
NEG	A ⇐ 00 - A																40	1	2																						50	1	2	60	2	7	70	3	6	X	X	X	X	X
DAA	A ⇐ Dez.korr.																19	1	2																													
DEC	A ⇐ A-1																4A	1	2																						5A	1	2	6A	2	7	7A	3	6	X	X	X	X	
EOR	A ⇐ A XOR M	88	2	2	98	2	3		A8	2	5		B8	3	4					C8	2	2		D8	2	3		E8	2	5		F8	3	4										X	X	0	X	0
INC	A ⇐ A+1																4C	1	2																						5C	1	2	6C	2	7	7C	3	6	X	X	X	X	
LDA	A ⇐ M	86	2	2	96	2	3		A6	2	5		B6	3	4					C6	2	2		D6	2	3		E6	2	5		F6	3	4										X	X	0	X	0
ORA	A ⇐ A ODER M	8A	2	2	9A	2	3		AA	2	5		BA	3	4					CA	2	2		DA	2	3		EA	2	5		FA	3	4										X	X	0	X	0
PSH	A ⇒ M SP ⇐ SP-1																36	1	4																						37	1	4										
PUL	SP ⇐ SP+1 A ⇐ M																32	1	4																						33	1	4										
ROL																	49	1	2																						59	1	2	69	2	7	79	3	6	X	X	X	X	X
ROR																	46	1	2																						56	1	2	66	2	7	76	3	6	X	X	X	X	X
ASL																	48	1	2																						58	1	2	68	2	7	78	3	6	X	X	X	X	X
ASR																	47	1	2																						57	1	2	67	2	7	77	3	6	X	X	X	X	X
LSR																	44	1	2																						54	1	2	64	2	7	74	3	6	X	X	0	X	X
STA	A ⇒ M				97	2	4		A7	2	6		B7	3	5									D7	2	4		E7	2	6		F7	3	5										X	X	0	X	0
SUB	A ⇐ A-M	80	2	2	90	2	3		A0	2	5		B0	3	4					C0	2	2		D0	2	3		E0	2	5		F0	3	4										X	X	X	X	X
SBA	A ⇐ A-B																10	1	2																													
SBC	A ⇐ A-M-C	82	2	2	92	2	3		A2	2	5		B2	3	4					C2	2	2		D2	2	3		E2	2	5		F2	3	4										X	X	X	X	X
TAB	B ⇐ A																16	1	2																													
TBA	A ⇐ B																17	1	2																													
TST	A - 00																4D	1	2																						5D	1	2	6D	2	7	7D	3	6	X	X	0	X	0

Adressierung →	unmitt.				zero-p.				indizl.				absolut				code-e.			Bedingungen					
Code	Wirkung	Op	B	Z	Op	B	Z		Op	B	Z		Op	B	Z		Op	B	Z	↕	H	I	N	Z	V	C
CPX	X - M	8C	3	3	9C	2	4		AC	2	6		BC	3	5											
DEX	X ⇐ X-1																09	1	4					X		
DES	SP ⇐ SP-1																34	1	4							
INX	X ⇐ X+1																08	1	4					X		
INS	SP ⇐ SP+1																31	1	4							
LDX	X ⇐ M	CE	3	3	DE	2	4		EE	2	6		FE	3	5									X	0	0
LDS	SP ⇐ M	8E	3	3	9E	2	4		AE	2	6		BE	3	5									X	0	0
STX	X ⇒ M				DF	2	5		EF	2	7		FF	3	6									X	0	0
STS	SP ⇒ M				9F	2	5		AF	2	7		BF	3	6											
TXS	SP ⇐ X-1																35	1	4							
TSX	X ⇐ SP+1																30	1	4							

Adressierung →	code-e.			Bedingungen						
Code	Wirkung	Op	B	Z	↕	H	I	N	Z	V	C
NOP	keine	01	1	2							
RTI	Rück. Int.	3B	1	10	vom Stapel						
RTS	Rück. Upro	39	1	5							
SWI	Softw. Int.	3F	1	12			1				
WAI	warte Int.	3E	1	9			1				
CLC	C ⇐ 0	0C	1	2							0
CLI	I ⇐ 0	0E	1	2			0				
CLV	V ⇐ 0	0A	1	2						0	
SEC	C ⇐ 1	0D	1	2							1
SEI	I ⇐ 1	0F	1	2			1				
SEV	V ⇐ 1	0B	1	2						1	
TAP	A ⇒ Bed.reg.	06	1	2	wie Akku						
TPA	A ⇐ Bed.reg.	07	1	2							

Adressierung →	relativ			Wirkung
Code	Bedingung	Op	B	Z	
BRA	keine	20	2	4	verzweige immer
BCC	C=0	24	2	4	verzweige bei Übertrag
BCS	C=1	25	2	4	verzweige bei Null
BEQ	Z=1	27	2	4	verzweige bei ≥ Null
BGE	N⊕V=0	2C	2	4	verzweige bei > Null
BGT	Z+(N⊕V)=0	2E	2	4	verzweige bei grösser
BHI	C+Z=0	22	2	4	verzweige bei ≤ Null
BLE	Z+(N⊕V)=1	2F	2	4	verzweige bei kleiner gleich
BLS	C+Z=1	23	2	4	verzweige bei < Null
BLT	N⊕V=1	2D	2	4	verzweige bei negativ
BMI	N=1	2B	2	4	verzweige bei ungleich Nul
BNE	Z=0	26	2	4	verzweige bei V=0
BVC	V=0	28	2	4	verzweige bei Überlauf
BVS	V=1	29	2	4	verzweige bei positiv
BPL	N=0	2A	2	4	verzweige bei Unterprogrammverzweigung
BSR	keine	8D	2	8	

Adressierung →	indizl.			absolut		
Code	Wirkung	Op	B	Z	Op	B	Z
JMP	springe	6E	2	4	7E	3	3
JSR	Unterpro.	AD	2	8	BD	3	9

Anhang 6: Befehlsliste des Motorola 6800

Adressierung →	Akkumulator A: unmitt.				zero-p.				indizl.				absolut				code-e.			Akkumulator B: unmitt.				zero-p.				indizl.				absolut				code-e.			Speicher: indizl.				absolut				Bedingungen	
Code	Wirkung	Op	B	Z	Op	B	Z		Op	B	Z		Op	B	Z		Op	B	Z	Op	B	Z		Op	B	Z		Op	B	Z		Op	B	C		Op	B	Z	Op	B	Z	Op	B	C	↕	HNZVC
ADD	A ⇐ A+M	8B	2	2	9B	2	3		AB	2	5		BB	3	4					CB	2	2		DB	2	3		EB	2	5		FB	3	4											X	XXXXX
ABA	A ⇐ A+B																1B	1	2																									X	XXXXX
ADC	A ⇐ A+M+C	89	2	2	99	2	3		A9	2	5		B9	3	4					C9	2	2		D9	2	3		E9	2	5		F9	3	4												XXXX
AND	A ⇐ A UND M	84	2	2	94	2	3		A4	2	5		B4	3	4					C4	2	2		D4	2	3		E4	2	5		F4	3	4												XXO
BIT	A UND M	85	2	2	95	2	3		A5	2	5		B5	3	4					C5	2	2		D5	2	3		E5	2	5		F5	3	4												XXO
CLR	A ⇐ 00																4F	1	2																										0100
CMP	A - M	81	2	2	91	2	3		A1	2	5		B1	3	4					C1	2	2		D1	2	3		E1	2	5		F1	3	4												XXXX
CBA	A - B																11	1	2																										XXXX
COM	A ⇐ NICHT A																43	1	2																53	1	2	63	2	7	73	3	6		XXO1
NEG	A ⇐ 00 - A																40	1	2																50	1	2	60	2	7	70	3	6		XXXX
DAA	A ⇐ Dez.korr.																19	1	2																										XXX
DEC	A ⇐ A-1																4A	1	2																5A	1	2	6A	2	7	7A	3	6		XXO
EOR	A ⇐ A XOR M	88	2	2	98	2	3		A8	2	5		B8	3	4					C8	2	2		D8	2	3		E8	2	5		F8	3	4												XXX
INC	A ⇐ A+1																4C	1	2																5C	1	2	6C	2	7	7C	3	6		XXO
LDA	A ⇐ M	86	2	2	96	2	3		A6	2	5		B6	3	4					C6	2	2		D6	2	3		E6	2	5		F6	3	4												XXO
ORA	A ⇐ A ODER M	8A	2	2	9A	2	3		AA	2	5		BA	3	4					CA	2	2		DA	2	3		EA	2	5		FA	3	4												XXO
PSH	A ⇒ M SP ⇐ SP-1																36	1	4																37	1	4										
PUL	SP ⇐ SP+1 A ⇐ M																32	1	4																33	1	4										
ROL																	49	1	2																59	1	2	69	2	7	79	3	6		XXXX
ROR																	46	1	2																56	1	2	66	2	7	76	3	6		XXXX
ASL																	48	1	2																58	1	2	68	2	7	78	3	6		XXXX
ASR																	47	1	2																57	1	2	67	2	7	77	3	6		XXXX
LSR																	44	1	2																54	1	2	64	2	7	74	3	6		XXO
STA	A ⇒ M				97	2	4		A7	2	6		B7	3	5									D7	2	4		E7	2	6		F7	3	5												XXO
SUB	A ⇐ A-M	80	2	2	90	2	3		A0	2	5		B0	3	4					C0	2	2		D0	2	3		E0	2	5		F0	3	4												XXXX
SBA	A ⇐ A-B																10	1	2																										XXXX
SBC	A ⇐ A-M-C	82	2	2	92	2	3		A2	2	5		B2	3	4					C2	2	2		D2	2	3		E2	2	5		F2	3	4												XXXX
TAB	B ⇐ A																16	1	2																										XXO
TBA	A ⇐ B																17	1	2																										XXO
TST	A - 00																4D	1	2																5D	1	2	6D	2	7	7D	3	6		XXOO

Adressierung →	unmitt.				zero-p.				indizl.				absolut				code-e.			Bedingungen	
Code	Wirkung	Op	B	Z	Op	B	Z		Op	B	Z		Op	B	Z		Op	B	Z	↕	HNZVC
CPX	X - M	8C	3	3	9C	2	4		AC	2	6		BC	3	5						XXX
DEX	X ⇐ X-1																09	1	4		X
DES	SP ⇐ SP-1																34	1	4		
INX	X ⇐ X+1																08	1	4		X
INS	SP ⇐ SP+1																31	1	4		
LDX	X ⇐ M	CE	3	3	DE	2	4		EE	2	6		FE	3	5						XXO
LDS	SP ⇐ M	8E	3	3	9E	2	4		AE	2	6		BE	3	5						XXO
STX	X ⇒ M				DF	2	5		EF	2	7		FF	3	6						XXO
STS	SP ⇒ M				9F	2	5		AF	2	7		BF	3	6						
TXS	SP ⇐ X-1																35	1	4		
TSX	X ⇐ SP+1																30	1	4		

Adressierung →	code-e.			Bedingungen	
Code	Wirkung	Op	B	Z	↕	HNZVC
NOP	keine	01	1	2		
RTI	Rück. int.	3B	1	10	vom Stapel	
RTS	Rück. Upro	39	1	5		
SWI	Softw.int.	3F	1	12		1
WAI	warte int.	3E	1	9		1
CLC	C ⇐ 0	0C	1	2		0
CLI	I ⇐ 0	0E	1	2	0	
CLV	V ⇐ 0	0A	1	2		0
SEC	C ⇐ 1	0D	1	2		1
SEI	I ⇐ 1	0F	1	2	1	
SEV	V ⇐ 1	0B	1	2		1
TAP	A ⇒ Bed.reg.	06	1	2	wie Akku	
TPA	A ⇐ Bed.reg.	07	1	2		

Adressierung →	relativ			
Code	Bedingung	Op	B	Z	Wirkung
BRA	keine	20	2	4	verzweige immer
BCC	C=0	24	2	4	verzweige bei C=0
BCS	C=1	25	2	4	verzweige bei Übertrag
BEQ	Z=1	27	2	4	verzweige bei Null
BGE	N⊕V=0	2C	2	4	verzweige bei ≥ Null
BGT	Z+(N⊕V)=0	2E	2	4	verzweige bei > Null
BHI	C+Z=0	22	2	4	verzweige bei grösser
BLE	Z+(N⊕V)=1	2F	2	4	verzweige bei ≤ Null
BLS	C+Z=1	23	2	4	verzweige bei kleiner gleich
BLT	N⊕V=1	2D	2	4	verzweige bei < Null
BMI	N=1	2B	2	4	verzweige bei negativ
BNE	Z=0	26	2	4	verzweige bei ungleich Nul
BVC	V=0	28	2	4	verzweige bei V=0
BVS	V=1	29	2	4	verzweige bei Überlauf
BPL	N=0	2A	2	4	verzweige bei positiv
BSR	keine	8D	2	8	Unterprogrammverzweigung

Adressierung →	indizl.			absolut		
Code	Wirkung	Op	B	Z	Op	B	Z
JMP	springe	6E	2	4	7E	3	3
JSR	Unterpro.	AD	2	8	BD	3	9

Anhang 6: Befehlsliste des Motorola 6800

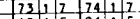
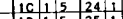
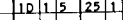
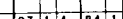
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		NOP code					TAP code	TPA code	INX code	DEX code	CLV code	SEV code	CLC code	SEC code	CLI code	SEI code
1	SBA code	CBA code					TAB code	TBA code		DAA code		ABA code				
2	BRA relativ		BHI relativ	BLS relativ	BCC relativ	BCS relativ	BNE relativ	BEQ relativ	BVC relativ	BVS relativ	BPL relativ	BMI relativ	BGE relativ	BLT relativ	BGT relativ	BLE relativ
3	TSX code	INS code	PUL A code	PUL B code	DES code	TXS code	PSH A code	PSH B code		RTS code		RTI code			WAI code	SWI code
4	NEG A code			COM A code	LSR A code		ROR A code	ASR A code	ASL A code	ROL A code	DEC A code		INC A code	TST A code		CLR A code
5	NEG B code			COM B code	LSR B code		ROR B code	ASR B code	ASL B code	ROL B code	DEC B code		INC B code	TST B code		CLR B code
6	NEG indiz.			COM indiz.	LSR indiz.		ROR indiz.	ASR indiz.	ASL indiz.	ROL indiz.	DEC indiz.		INC indiz.	TST indiz.	JMP indiz.	CLR indiz.
7	NEG absolut			COM absolut	LSR absolut		ROR absolut	ASR absolut	ASL absolut	ROL absolut	DEC absolut		INC absolut	TST absolut	JMP absolut	CLR absolut
8	SUB A unmitt.	CMP A unmitt.	SBC A unmitt.		AND A unmitt.	BIT A unmitt.	LDA A unmitt.		EOR A unmitt.	ADC A unmitt.	ORA A unmitt.	ADD A unmitt.	CPX unmitt.	BSR relativ	LDS unmitt	
9	SUB A zero-p.	CMP A zero-p.	SBC A zero-p.		AND A zero-p.	BIT A zero-p.	LDA A zero-p.	STA A zero-p.	EOR A zero-p.	ADC A zero-p.	ORA A zero-p.	ADD A zero-p.	CPX zero-p.		LDS zero-p.	STS zero-p.
A	SUB A indiz.	CMP A indiz.	SBC A indiz.		AND A indiz.	BIT A indiz.	LDA A indiz.	STA A indiz.	EOR A indiz.	ADC A indiz.	ORA A indiz.	ADD A indiz.	CPX indiz.	JSR indiz.	LDS indiz.	STS indiz.
B	SUB A absolut	CMP A absolut	SBC A absolut		AND A absolut	BIT A absolut	LDA A absolut	STA A absolut	EOR A absolut	ADC A absolut	ORA A absolut	ADD A absolut	CPX absolut	JSR absolut	LDS absolut	STS absolut
C	SUB B unmitt.	CMP B unmitt.	SBC B unmitt.		AND B unmitt.	BIT B unmitt.	LDA B unmitt.		EOR B unmitt.	ADC B unmitt.	ORA B unmitt.	ADD B unmitt.			LDX unmitt.	
D	SUB B zero-p.	CMP B zero-p.	SBC B zero-p.		AND B zero-p.	BIT B zero-p.	LDA B zero-p.	STA B zero-p.	EOR B zero-p.	ADC B zero-p.	ORA B zero-p.	ADD B zero-p.			LDX zero-p.	STX zero-p.
E	SUB B indiz.	CMP B indiz.	SBC B indiz.		AND B indiz.	BIT B indiz.	LDA B indiz.	STA B indiz.	EOR B indiz.	ADC B indiz.	ORA B indiz.	ADD B indiz.			LDX indiz.	STX indiz.
F	SUB B absolut	CMP B absolut	SBC B absolut		AND B absolut	BIT B absolut	LDA B absolut	STA B absolut	EOR B absolut	ADC B absolut	ORA B absolut	ADD B absolut			LDX absolut	STX absolut

96 [illegible]

Anhang 7: Befehlsliste des Intel 8080/8085

Code	Wirkung	unmitt. Op	B	Z	Akku A Op	B	Z	B-Reg. Op	B	Z	C-Reg. Op	B	Z	D-Reg. Op	B	Z	E-Reg. Op	B	Z	H-Reg. Op	B	Z	L-Reg. Op	B	Z	M-Spei. Op	B	Z	Bedingungen NZ-H	-P	-C
MVI	r ⇐konst.				3E	2	10	06	2	10	0E	2	10	16	2	10	1E	2	10	26	2	10	2E	2	10	36	2	10			
MOV	A ⇐reg.				7F	1	5	78	1	5	79	1	5	7A	1	5	7B	1	5	7C	1	5	7D	1	5	7E	1	7			
MOV	B ⇐reg.				47	1	5	40	1	5	41	1	5	42	1	5	43	1	5	44	1	5	45	1	5	46	1	7			
MOV	C ⇐reg.				4F	1	5	48	1	5	49	1	5	4A	1	5	4B	1	5	4C	1	5	4D	1	5	4E	1	7			
MOV	D ⇐reg.				57	1	5	50	1	5	51	1	5	52	1	5	53	1	5	54	1	5	55	1	5	56	1	7			
MOV	E ⇐reg.				5F	1	5	58	1	5	59	1	5	5A	1	5	5B	1	5	5C	1	5	5D	1	5	5E	1	7			
MOV	H ⇐reg.				67	1	5	60	1	5	61	1	5	62	1	5	63	1	5	64	1	5	65	1	5	66	1	7			
MOV	L ⇐reg.				6F	1	5	68	1	5	69	1	5	6A	1	5	6B	1	5	6C	1	5	6D	1	5	6E	1	7			
MOV	M ⇐reg.				77	1	7	70	1	7	71	1	7	72	1	7	73	1	7	74	1	7	75	1	7						
INR	reg ⇐reg +1				3C	1	5	04	1	5	0C	1	5	14	1	5	1C	1	5	24	1	5	2C	1	5	34	1	10	XX	X		X	
DCR	reg ⇐reg -1				3D	1	5	05	1	5	0D	1	5	15	1	5	1D	1	5	25	1	5	2D	1	5	35	1	10	XX	X		X	
ADI	A ⇐A+konst	C6	2	7																									XX	X		X	X
ADD	A ⇐A+reg.				87	1	4	80	1	4	81	1	4	82	1	4	83	1	4	84	1	4	85	1	4	86	1	7	XX	X		X	X
ACI	A ⇐A+konst+C	CE	2	7																									XX	X		X	X
ADC	A ⇐A+reg.+C				8F	1	4	88	1	4	89	1	4	8A	1	4	8B	1	4	8C	1	4	8D	1	4	8E	1	7	XX	X		X	X
SUI	A ⇐A-konst	D6	2	7																									XX	X		X	X
SUB	A ⇐A-reg.				97	1	4	90	1	4	91	1	4	92	1	4	93	1	4	94	1	4	95	1	4	96	1	7	XX	X		X	X
SBI	A ⇐A-konst-C	DE	2	7																									XX	X		X	X
SBB	A ⇐A-reg.-C				9F	1	4	98	1	4	99	1	4	9A	1	4	9B	1	4	9C	1	4	9D	1	4	9E	1	7	XX	X		X	X
ANI	A ⇐A UND kon.	E6	2	7																									XX	X		X	X
ANA	A ⇐A UND reg.				A7	1	4	A0	1	4	A1	1	4	A2	1	4	A3	1	4	A4	1	4	A5	1	4	A6	1	7	XX	X		X	X
ORI	A ⇐A ODER kon	F6	2	7																									XX	X		X	X
ORA	A ⇐A ODER reg				B7	1	4	B0	1	4	B1	1	4	B2	1	4	B3	1	4	B4	1	4	B5	1	4	B6	1	7	XX	X		X	X
XRI	A ⇐A XOR kon.	EE	2	7																									XX	X		X	X
XRA	A ⇐A XOR reg.				AF	1	4	A8	1	4	A9	1	4	AA	1	4	AB	1	4	AC	1	4	AD	1	4	AE	1	7	XX	X		X	X
CPI	A - konst.	FE	2	7																									XX	X		X	X
CMP	A - reg.				BF	1	4	B8	1	4	B9	1	4	BA	1	4	BB	1	4	BC	1	4	BD	1	4	BE	1	7	XX	X		X	X

Akkumulator

Code	Wirkung	Op	B	Z	Bedingungen NZ	-H	-P	-C
LDA	A ⇐Speicher	3A	3	13				
LDAXB	A ⇐Sp. (B,C)$_{hl}$	0A	1	7				
LDAXD	A ⇐Sp. (D,E)$_{hl}$	1A	1	7				
IN	A ⇐Eingabe	DB	2	10				
STA	A⇒ Speicher	32	3	13				
STAXB	A⇒ Sp. (B,C)$_{hl}$	02	1	7				
STAXD	A⇒ Sp. (D,E)$_{hl}$	12	1	7				
OUT	A⇒ Ausgabe	D3	2	10				
RLC		07	1	4				X
RRC		0F	1	4				X
RAL		17	1	4				X
RAR		1F	1	4				X
CMA	A ⇐ NICHT A	2F	1	4				
DAA	A ⇐Dez.korr.	27	1	4	XX	X	X	X
XRA A	A ⇐00,C ⇐00	AF	1	4	01	0	0	0

Code	Wirkung	Op	B	Z
ORA A	C ⇐0	B7	1	4
STC	C ⇐1	37	1	4
CMC	C ⇐ NICHT C	3F	1	4
EI	Int. frei	FB	1	4
DI	Int. gesp.	F3	1	4
HLT	anhalten	76	1	7
NOP	keine	00	1	4
RIM	nur 8085	20	1	4
SIM	nur 8085	30	1	4

Registerpaar

Code	Wirkung	B (B,C) Op	B	Z	D (D,E) Op	B	Z	H (H,L) Op	B	Z	SP Op	B	Z	PSW ,A Op	B	Z	Bedingungen NZ-H	-P	-C
LXI	rp ⇐konst.	01	3	10	11	3	10	21	3	10	31	3	10						
INX	rp ⇐rp+1	03	1	5	13	1	5	23	1	5	33	1	5						
DCX	rp ⇐rp-1	0B	1	5	1B	1	5	2B	1	5	3B	1	5						
DAD	H,L ⇐H,L+rp	09	1	10	19	1	10	29	1	10	39	1	10						X
PUSH	rp⇐ Stapel	C5	1	10	D5	1	10	E5	1	10				F5	1	10			
POP	rp ⇐Stapel	C1	1	10	D1	1	10	E1	1	10				F1	1	10			

Registerpaar H,L

Code	Wirkung	Op	B	Z
LHLD	H,L ⇐Speicher	2A	3	16
SHLD	H,L⇒ Speicher	22	3	16
XCHG	H,L ⇔ D,E	EB	1	4
XTHL	H,L ⇔ Stapel	E3	1	18
PCHL	H,L⇒ Bef.zäh.	E9	1	5
SPHL	H,L⇒ SP	F9	1	5

Sprungbefehle

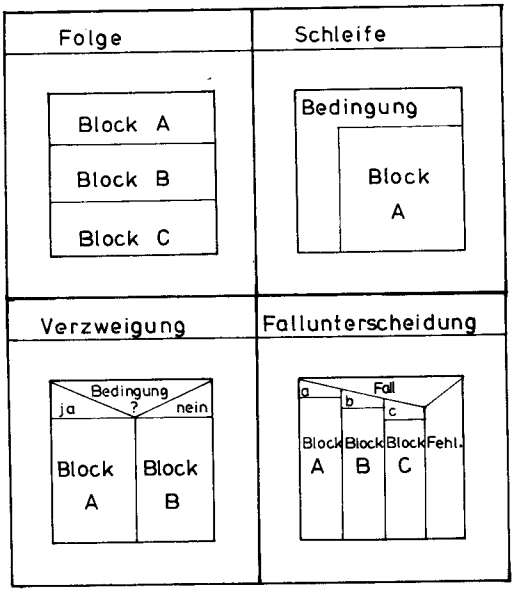
Bedingung	Sprung Code	Op	B	Z	Unterprogramm Code	Op	B	Z	Rücksprung Code	Op	B	Z
keine	JMP	C3	3	10	CALL	CD	3	17	RET	C9	1	10
Z = 1	JZ	CA	3	10	CZ	CC	3	11/17	RZ	C8	1	5/11
Z = 0	JNZ	C2	3	10	CNZ	C4	3	11/17	RNZ	C0	1	5/11
C = 1	JC	DA	3	10	CC	DC	3	11/17	RC	D8	1	5/11
C = 0	JNC	D2	3	10	CNC	D4	3	11/17	RNC	D0	1	5/11
N = 1	JM	FA	3	10	CM	FC	3	11/17	RM	F8	1	5/11
N = 0	JP	F2	3	10	CP	F4	3	11/17	RP	F0	1	5/11
P = 1	JPE	EA	3	10	CPE	EC	3	11/17	RPE	E8	1	5/11
P = 0	JPO	E2	3	10	CPO	E4	3	11/17	RPO	E0	1	5/11

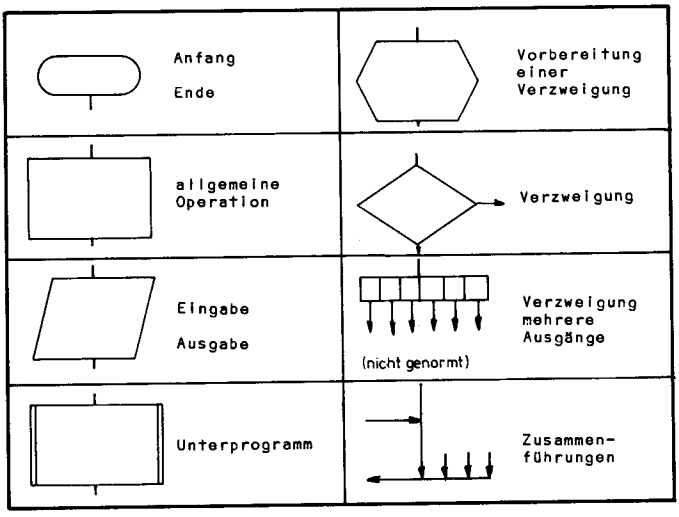
RST-Befehle

Code	Op	B	Z	Adresse
RST 0	C7	1	11	0000
RST 1	CF	1	11	0008
RST 2	D7	1	11	0010
RST 3	DF	1	11	0018
RST 4	E7	1	11	0020
RST 5	EF	1	11	0028
RST 6	F7	1	11	0030
RST 7	FF	1	11	0038

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI B,	STAX B	INX B	INR B	DCR B	MVI B,	RLC		DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,	RRC
1		LXI D,	STAX D	INX D	INR D	DCR D	MVI D,	RAL		DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,	RAR
2	RIM	LXI H,	SHLD	INX H	INR H	DCR H	MVI H,	DAA		DAD H	LHLD	DCX H	INR L	DCR L	MVI L,	CMA
3	SIM	LXI S,	STA	INX S	INR M	DCR M	MVI M,	STC		DAD SP	LDA	DCX SP	INR A	DCR A	MVI A,	CMC
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
C	RNZ	POP B	JNZ	JMP	CNZ	PUSH B	ADI	RST 0	RZ	RET	JZ		CZ	CALL	ACI	RST 1
D	RNC	POP D	JNC	OUT	CNC	PUSH D	SUI	RST 2	RC		JC	IN	CC		SBI	RST 3
E	RPO	POP H	JPO	XTHL	CPO	PUSH H	ANI	RST 4	RPE	PCHL	JPE	XCHG	CPE		XRI	RST 5
F	RP	POP PSW	JP	DI	CP	PUSH PS	ORI	RST 6	RM	SPHL	JM	EI	CM		CPI	RST 7

Anhang 8: Sinnbilder für Programmablaufpläne und Programmblockpläne (Struktogramme)





11. Register

11.1 Allgemeiner Teil

Ablaufsteuerung 36
Addition 21
Additionsbefehl 40
Adreßleitung 37, 39, 42
Adreßregister 42
Akkumulator 33
ALU 33
AND-Schaltung 19, 207
Antivalenz 20, 208
Äquivalenz 20
Arithmetische Befehle 40, 222
Arithmetisches Schieben 216
Arithmetisch-Logische Einheit 33
ASCII-Code 13, 336
Assembler 47, 66

Basis-Flip-Flop 29, 38
Baumdiagramm 288
BCD-Code 13, 23, 223
Bedingter Sprung 43, 115
Bedingungsregister 33, 43, 115
Befehlsablauf 36, 106
Befehlsdecoder 42
Befehlsliste 46
Befehlsregister 43, 107
Befehlsschaltwerk 43
Befehlszählregister 42, 107
Berechneter Sprung 197
Betriebssystem 51
Binär 12
Binäres Suchen 272
Bit 13
Bus 40
Byte 13

Carry-Bit 22
Code-eigene Adressierung 75
Cross-Assembler 66

Datenregister 95
Datentransport 40, 75, 93, 106, 142
Dezimalkorrektur 24, 223
Dezimalzahl 13, 14
D-Flip-Flop 31
Dividieren 217, 259
Divisions-Rest-Verfahren 15
Dualzahl 13, 14

Ein/Ausgaberegister 39, 95
Eklusives ODER 20, 208

Fallabfrage 171
Flip-Flop 29
Folge 94, 106

Halbaddierer 20
Hexadezimalzahl 16
H-Potential 12

Indexregister 46, 142
Indirekte Adressierung 163
Indirekter Sprung 197
Interrupt 44, 47, 298

JA-Schaltung 19

Komplement 17, 20, 206
Konstanten 44, 67, 81
Kontrollregister 95
Korrekturaddierer 24

Lade-Befehl 40, 75, 80
Logische Befehle 41, 185, 206
L-Potential 12

Maske 185, 207
Master-Slave 31
Multiplizieren 216, 255

NAND-Basis-Flip-Flop 29
NAND-Schaltung 20
N-Bit 23
Negative Zahlen 17
NICHT-Schaltung 19, 206
NOR-Basis-Flip-Flop 38
NOR-Schaltung 20
NOT-Schaltung 19, 206
Nur-Lese-Speicher 37

ODER-Schaltung 19, 207
Oktalzahl 16
OR-Schaltung 19, 207
Overflow 23

Paralleladdierer 22
Parallelschnittstelle 52, 95
Programmablaufplan 47, 64, 93
Programmblockplan 64, 94
Programmspeicher 42
Programmverzweigung 171

RAM-Speicher 37
Rechenschaltung 19
Rechenwerk 39
Reelle Zahl 248
Reset 44, 298
Richtungsregister 95
Ringzähler 35
ROM-Speicher 37
Rotieren 215
RS-Flip-Flop 30
Rücksprung 44, 291
Rückwärtssprung 109

Schaltwerk 35
Schiebebefehl 33, 41, 188, 215
Schiebeoperation 41, 215
Schieberegister 32
Schleife 106, 109, 117
Schreib-Lese-Speicher 37
Sortieren 269
Speicherbefehl 40, 75, 89
Speicherschaltung 29
Sprungbefehl 43, 106, 172
Stapel 46, 276
Steuerregister 95
Steuerwerk 42
Subtraktion 25
Subtraktionsbefehl 40

Tabellen 199, 268
Tabellensuchen 269
Takt 30

Überlauf 23
Übertrag 22
Unbedingter Sprung 43, 108
UND-Schaltung 19, 207
Unmittelbare Adressierung 81
Unterlauf 23
Unterprogramm 44, 262, 276, 287

V-Bit 23
Vergleichsbefehle 127, 172
Verzweigung 117, 171
Volladdierer 21

Vorübertrag 22
Vorwärtssprung 109, 116

XOR-Schaltung 20, 208

Zählbefehl 125
Zähler 33, 125
Zählschleife 125
Z-Bit 25

11.2 KIM-1-Teil

Absolute Adressierung 81
ADC-Befehl 225
Additionsbefehl 224
Adreßplan 54
Adreßzähler 68
Akkumulator 75
AND-Befehl 186, 209
Anschlüsse 53
Arithmetische Befehle 223
ASLA-Befehl 218
ASL-Befehl 218
Assembler 68

B-Bit 83, 300
BCC-Befehl 119
BCD-Zahlen 226, 229, 250
BCS-Befehl 119
Bedingter Sprung 117
Bedingungsregister 83
BEQ-Befehl 119
BIT-Befehl 173, 210
BMI-Befehl 119
BNE-Befehl 119
BPL-Befehl 119
BRK-Befehl 300
BVC-Befehl 119
BVS-Befehl 119
BYTE-Anweisung 69

C-Bit 83
CLC-Befehl 84, 225
CLD-Befehl 84, 223
CLI-Befehl 84, 300
CLV-Befehl 84
CMP-Befehl 131, 145, 165, 167
CPX-Befehl 131
CPY-Befehl 131

Datenregister 96
D-Bit 83, 223
DEC-Befehl 129, 145
DEX-Befehl 129
DEY-Befehl 129
Dezimalarithmetik 223
Dezimalkorrektur 223
Division 263
Dualarithmetik 223

End-Anweisung 69
EOR-Befehl 211

I-Bit 83
INC-Befehl 129, 145
Indexregister 143
Indirekte Adressierung 164
Indirekter Sprung 200
Indirekt-indizierte Adressierung 166
Indizierte Adressierung 144
Indiziert-indirekte Adressierung 164
Interrupt 299
INX-Befehl 129
INY-Befehl 129
IRQ-Interrupt 299, 302

JMP-Befehl 110, 200
JSR-Befehl 292

Lade-Befehl 81, 145, 165, 167
LDA-Befehl 82, 145, 165, 167
LDX-Befehl 82, 145
LDY-Befehl 82, 145
Logische Befehle 186, 209
LSRA-Befehl 218
LSR-Befehl 218

Maske 209
Multiplikation 262, 279

N-Bit 83
NMI-Interrupt 299, 302
NOP-Befehl 130

ORA-Befehl 186, 210

Parallelschnittstelle 96
PHA-Befehl 278
PHP-Befehl 278
PLA-Befehl 278
PLP-Befehl 278

Register 53, 54, 75
Relative Adressierung 117
RESET 96, 299
Richtungsregister 96
ROLA-Befehl 191, 217
ROL-Befehl 217
RORA-Befehl 191, 217
ROR-Befehl 217
RTI-Befehl 300
RTS-Befehl 292

SBC-Befehl 227
Schiebebefehle 191, 217
SEC-Befehl 84, 227
SED-Befehl 84, 223
SEI-Befehl 84, 300
Seite 119, 144
Speicherbefehle 89, 145, 165, 167
Sprungweite 117
STA-Befehl 89, 145, 165, 167
Stapel 277, 293, 300
Stapelbefehle 277
Stapelzeiger 277, 293, 300
STX-Befehl 89, 145
STY-Befehl 89, 145
Subtraktionsbefehle 227

TAX-Befehl 76
TAY-Befehl 76
TSX-Befehl 76
TXA-Befehl 76
TXS-Befehl 76
TYA-Befehl 76

Unbedingter Sprung 110
Unmittelbare Adressierung 83
Unterprogramm 262, 292

Variablenreservierung 69
V-Bit 83
Vergleichsbefehle 131, 145, 165, 167, 173
Verzweigungen 173, 186, 191

WORD-Anweisung 70

X-Register 75, 143, 249

Y-Register 75, 143, 249

Z-Bit 83
Zählbefehl 129, 145
Zählschleife 130
Zero-page Adressierung 82

11.3 6800-Teil

ABA-Befehl 231
Absolute Adressierung 84
ADC-Befehl 231
ADD-Befehl 231
Adreßkonstante 151
Adreßplan 58
AND-Befehl 187, 211
Anschlüsse 56
Arithmetische Befehle 229
ASL-Befehl 220
ASR-Befehl 220
Assembler 70

BCC-Befehl 121, 122
BCD-Zahlen 236, 252
BCS-Befehl 121, 122
Bedingter Sprung 121
BEQ-Befehl 121, 122
BGE-Befehl 122
BGT-Befehl 122
BHI-Befehl 122
BIT-Befehl 212
BLE-Befehl 122
BLS-Befehl 122
BLT-Befehl 122
BMI-Befehl 121
BNE-Befehl 121
BPL-Befehl 121
BRA-Befehl 112, 114
BSR-Befehl 294
BVC-Befehl 121
BVS-Befehl 121

CBA-Befehl 136
C-Bit 86
CLC-Befehl 86, 230
CLI-Befehl 86, 304
CLR-Befehl 86
CLV-Befehl 86
CMP-Befehl 136, 151
COM-Befehl 211
CPX-Befehl 136, 151

DAA-Befehl 236
Datenregister 98
DEC-Befehl 132, 151
DES-Befehl 132
DEX-Befehl 132
Dezimalkorrektur 236
Direkte Adressierung 84
Division 265

END-Anweisung 71
EOR-Befehl 213
EQU-Anweisung 71
Extended Adressierung 84

FCB-Anweisung 71
FDB-Anweisung 72

H-Bit 86, 229 236

I-Bit 86, 304
INC-Befehl 132, 151
Indexed Adressierung 150, 168, 201
Indexregister 133, 149, 168, 201
Indirekte Adressierung 167
Indirekter Sprung 201
Indizierte Adressierung 150, 201, 294
INS-Befehl 132
Interrupt 303
INX-Befehl 132
IRQ-Interrupt 304, 307

JMP-Befehl 111, 201
JSR-Befehl 294

Kontrollregister 98

Ladebefehle 84, 150
LDA-Befehl 84, 150
LDS-Befehl 84, 150
LDX-Befehl 84, 150
Logische Befehle 187, 211
LSR-Befehl 220

Maske 211
Multiplikation 265, 282

N-Bit 86
NEG-Befehl 234
NMI-Interrupt 303, 307
NOP-Befehl 134

ORA-Befehl 187, 212
ORG-Anweisung 70

Parallelschnittstelle 98, 308
PSH-Befehl 280
PUL-Befehl 280

Register 56, 57, 77
Relative Adressierung 112
RESET 98, 303
Richtungsregister 98
RMB-Anweisung 71
ROL-Befehl 193, 219
ROR-Befehl 193, 219
RTI-Befehl 305
RTS-Befehl 294

SBA-Befehl 233
SBC-Befehl 234
Schiebebefehle 193, 219
SEC-Befehl 86, 230
SEI-Befehl 86, 304
SEV-Befehl 86
Speicherbefehle 90, 150
Sprungweite 112
STA-Befehl 91, 150
Stapel 280, 295, 305
Stapelbefehle 280
Stapelzeiger 280, 295, 305
STS-Befehl 91, 150
STX-Befehl 91, 150
SUB-Befehl 233
Subtraktion 233
SWI-Befehl 304, 307

TAB-Befehl 77
TAP-Befehl 77
TBA-Befehl 77
TPA-Befehl 77
TST-Befehl 177
TSX-Befehl 77
TXS-Befehl 77

Unbedingter Sprung 111
Unmittelbare Adressierung 85
Unterprogramme 265, 294

V-Bit 86
Vergleichsbefehle 136, 151, 177
Verzweigungen 177, 187, 193

WAI-Befehl 304, 307

X-Register 133, 149, 168, 201

Z-Bit 86
Zählbefehle 132, 151
Zero-page Adressierung 85

11.4 8080/8085-Teil

Absolute Adressierung 87
ACI-Befehl 240
ADC-Befehl 240
ADD-Befehl 240
Additionsbefehle 239
ADI-Befehl 124, 240
Adreßplan 61
ANA-Befehl 214
ANI-Befehl 188, 214
Arithmetische Befehle 238
Anschlüsse 60
Assembler 72

BCD-Zahlen 244, 254
Bedingter Sprung 123
Bedingungsregister 88

CALL-Befehl 296
C-Bit 88
CC-Befehl 296
CMA-Befehl 213
CM-Befehl 296
CMC-Befehl 88, 239
CMP-Befehl 140, 162
CNC-Befehl 296
CNZ-Befehl 296
CP-Befehl 296
CPE-Befehl 296
CPI-Befehl 140
CPO-Befehl 296
CZ-Befehl 296

DAA-Befehl 245
DAD-Befehl 240, 253
Datenregister 104
DB-Anweisung 73
DCR-Befehl 137, 162
DCX-Befehl 137, 254
Dezimalkorrektur 245
DI-Befehl 310
Direkte Adressierung 87
Division 268
DS-Anweisung 73
DW-Anweisung 74

EI-Befehl 310
END-Anweisung 73
EQU-Anweisung 73

H-Bit 88, 238
HLT-Befehl 311

IN-Befehl 102
Indexregister 157
Indirekte Adressierung 169
Indirekter Sprung 203
INR-Befehl 137, 162
Interrupt 309, 312
INX-Befehl 137, 254

JC-Befehl 123
JM-Befehl 123
JMP-Befehl 115
JNC-Befehl 123
JNZ-Befehl 123
JP-Befehl 123
JPE-Befehl 123
JPO-Befehl 123
JZ-Befehl 123

Ladebefehl 87, 159, 161
LDA-Befehl 87
LDAX-Befehl 159
LHLD-Befehl 87, 153
Logische Befehle 188, 213
LXI-Befehl 88, 254

Maske 214
MOV-Befehl 78, 161, 162
Multiplikation 266, 286
MVI-Befehl 88, 162

N-Bit 88
NOP-Befehl 138

ORA-Befehl 214
ORG-Anweisung 72
ORI-Befehl 188, 214
OUT-Befehl 102

Parallelschnittstelle 62, 100, 103, 104
P-Bit 88
PCHL-Befehl 79, 203, 254
POP-Befehl 254, 284
PSW-Register 284
PUSH-Befehl 254, 284

RAL-Befehl 195, 221
RAR-Befehl 195, 221
RC-Befehl 296
Register 60, 61, 78
Registerpaar-Adressierung 157, 169, 203
Reset 100, 104, 309, 312
RET-Befehl 296
Richtungsregister 104
RIM-Befehl 312
RLC-Befehl 221
RM-Befehl 296
RNC-Befehl 296
RNZ-Befehl 296
RP-Befehl 296
RPE-Befehl 296
RPO-Befehl 296
RRC-Befehl 221
RST-Befehl 310
RZ-Befehl 296

SBB-Befehl 242
SBI-Befehl 242
Schiebebefehl 195, 221
SHLD-Befehl 92, 253
SIM-Befehl 312
Speicherbefehl 92, 159 162
SPHL-Befehl 79, 254
STA-Befehl 92
Stapel 283, 297
Stapelbefehle 283
Stapelzeiger 241, 283, 297
STAX-Befehl 159
STC-Befehl 88, 239
Steuerregister 100
SUB-Befehl 243
Subtraktionsbefehl 242
SUI-Befehl 243

Unbedingter Sprung 114
Unterprogramme 262, 296

Vergleichsbefehle 140, 162, 181
Verzweigungen 181, 188, 195

XCHG-Befehl 79, 253
XRA-Befehl 215
XRI-Befehl 215
XTHL-Befehl 254, 285

Zählbefehle 137, 162
Z-Bit 88

Mikroprozessoren
Das aktuelle Fachbuch

Werner Kobitzsch

Mikroprozessoren – Aufbau und Wirkungsweise

Hier liegt ein ausführliches Lehrbuch vor. Es befaßt sich im Gegensatz zu üblichen Firmendarstellungen nicht mit speziellen Bauelementen, sondern zeigt die Grundlagen des Aufbaues und der Funktion von Mikroprozessoren. Ihre besonderen Probleme, die durch die hochintegrierte Bauweise bedingt sind (z.B. begrenzte Zahl von Anschlußleitungen und begrenzte Befehlsvorräte) werden schwerpunktmäßig behandelt.

Teil 1: Grundlagen

1977. 204 Seiten, 139 Abbildungen, 11 Tabellen, DM 38,– ISBN 3-486-20801-2

Aus dem Inhalt: Zahlensysteme: Darstellung und Grundrechenarten – Die Baugruppen eines Mikrocomputers – Überblick über Adressierungsarten – Befehlsvorrat eines hypothetischen Mikroprozessors – Der Mikroprozessor in Verbindung mit Peripheriegeräten – Softwarehilfsmittel zur Programmerstellung – Mikroprogrammierung: Definition und Grundlagen.

Teil 2: Beispiele und Anwendungen

1980. 480 Seiten, zahlreiche Abbildungen und Tabellen, DM 48,– ISBN 3-486-21611-2
Reihe: Datenverarbeitung

Im Mittelpunkt dieses Teils steht die anwendungsorientierte Information über typische Mikrocomputer und -prozessoren. Es ist besonders auf Architektur, Bussystem, Betriebsarten, Befehlsvorrat, Systembausteine und Programmbeispiele Wert gelegt.

Aus dem Inhalt: Überblick über Mikroprozessoren und Mikrocomputer – Beschreibung verschiedener Mikroprozessor- und Mikrocomputerfamilien (MCS-48, F 8, MCS-80/85, 6800, 9900) – Programmbeispiele zu Mikrocomputern und -prozessoren (8048, F 8, 8080 A/8085 A, 6800, 9900) – Gedanken zum Ablauf eines Mikroprozessorprojekts.

Oldenbourg

Mikroprozessoren
Das aktuelle Fachbuch

Mikroprozessoren und ihre Anwendungen 2

Einführung, heutiger Stand der Technik, Möglichkeiten und Probleme der Anwendungen, Zukunftsaussichten
Zweites Darmstädter Kolloquium

Herausgegeben von Wolfgang Hilberg und Robert Piloty
1979. 512 Seiten, 251 Abbildungen, 20 Tabellen, DM 68,— ISBN 3-486-23701-2

Die Zusammenfassung des neuesten Standes dieser revolutionären Technik! Führende Fachleute aus Industrie und Hochschulen geben mit ihren (inhaltlich aufeinander abgestimmten) Beiträgen einen aktuellen Überblick über das ganze Gebiet des Entwurfs, der Herstellung, der Handhabung und der Anwendung von Mikroprozessoren.

Aus dem Inhalt:

Einleitung: Vom Transistor zum System: Der Mikrocomputer; Grundsätzliche Funktionsweise von Mikroprozessoren; Mikroprozessoren und Mikrocomputer — Überblick, Wirtschaftlichkeit, Trends; Der Einsatz von Mikroprozessoren — Einfluß auf betriebliche Funktionsabläufe und Beschäftigungsstruktur.

Der Mikroprozessor: Grundsätzliche Funktionsweise von mikroprogrammierbaren Rechnern; Bit-Slice-Mikroprozessoren; Programmiermethodik, Stand und Trends; Kleinrechner und problemorientierte Programmierung; Moderne Großintegrationstechniken für Mikroprozessoren; Neuartige Herstellungstechniken für hochintegrierte Halbleiter-Bauelemente; Mikroprozessoren und Mikrocomputer — Entwicklungshilfsmittel; Anwendungssoftware bei Mikroprozessorprojekten; Einsatz des Mikroprozessors für Aufgaben im Rechnerverbund; Mikroprozessoren — Zuverlässigkeitsangaben und Testverfahren; Mikroprozessoren — Qualifikation, Prüftechnik, Qualitätskontrolle.

Anwendung von Mikroprozessoren/Mikrorechnern in: der Regelungstechnik; Biologie und Medizin; der Vermittlungstechnik, Hausgeräten, der Laborautomatisierung; der digitalen Signalverarbeitung; Fernmeldezentralen; im Kraftfahrzeug; Meßgeräten; graphisch-interaktiven Systemen; der Verarbeitung von Bildern; der Textverarbeitung; der Uhrentechnik; Funkuhren; einer privaten Mikrodatenbank.

Oldenbourg

Der Verfasser, Dipl.-Ing. Günter Schmitt, hat an der TU Berlin Elektrotechnik studiert und von 1963–1969 bei der Siemens AG in Berlin Meßwandler entwickelt. Seit 1969 ist er Fachhochschullehrer an der Fachhochschule der Deutschen Bundespost in Dieburg für die Fächer Programmieren und Datenverarbeitung, Maschinenorientierte Programmierung und Mikroprozessoren.

Das vorliegende Werk ist eine systematische und didaktisch aufgebaute Einführung in die maschinenorientierte Programmierung mit vollständigen Programmbeispielen für die wichtigsten Mikroprozessoren 6502 (KIM), 6800 und 8080. Die Einführung entwickelt einen einfachen Modell-Mikrocomputer. Es folgt eine Beschreibung von drei im Lehr- und Hobby-Bereich verbreiteten Systemen: KIM-1, MEK 6800 und SDK 85. Alle Beispiele und Übungen können mit diesen Systemen getestet werden. Im Hauptteil werden Programmierverfahren beschrieben: lineare Programme, Sprünge, Verzweigungen, Schleifen, Indizierung, Stapel, Unterprogramme und Interrupt. Jedes Verfahren wird für die Mikroprozessoren 6502, 6800 und 8080 durch Programmblockpläne und Assemblerprogramme erläutert. Zahlreiche Abbildungen zeigen die Vorgänge im Rechner. Für alle Aufgaben gibt es Lösungen.

ISBN 3-486-22892-7